Training Text

マイコントレーニングボード
MT-R300

グラフィックLCD基板（MT-E507）編

VPort Lab. 代表
元長野県工科短期大学校教授
工学博士 星野 俊行 著

Sunhayato

安全上のご注意

このたびは弊社製品をご使用いただき、誠にありがとうございます。本項では、誤った取り扱いによる事故を未然に防ぐための安全上の注意事項を説明しています。弊社製品をご使用になる前に必ずお読みください。

⚠警告	この表記を無視して誤った取り扱いをすると、死亡や重傷など、人体への重大な障害をもたらす恐れのある内容について示しています。
⚠注意	この表記を無視して誤った取り扱いをすると、軽傷または中程度の障害をもたらす恐れのある内容について示しています。また、本製品や本製品に接続している機器に損傷を与える可能性がある事項についても示しています。

⚠警告

水場禁止	▶水分の多いところ、水がかかる場所では、本製品は使用しないでください 風呂場や台所など水分の多いところ、水がかかる場所では、本製品は使用しないでください。 火災、感電、故障の原因となります。
禁止	▶医療、軍事、航空宇宙、列車、運送、原子力などの制御設備へは使用しないでください 医療機器、軍事機器、航空宇宙機器、運送、原子力などの制御設備などの人命に関わるシステムへの使用は意図しておりません。
接触禁止	▶雷が鳴りはじめたらご使用をお控えください 近くに雷が発生したときは、パソコンから本製品を抜いてご使用をお控えください。また、ご使用のパソコンの取扱説明書に従って、電源プラグをコンセントから抜くなどしてください。雷によっては、火災、感電、故障の原因となることがあります。

⚠注意

プラグの差し込み	▶プラグは確実に差し込んでください 差し込みが不完全ですと火災、感電、過熱、故障の原因になります。
発火注意	▶発火、発煙、異臭への対処 発火、発煙、異臭がするなどの異常がありましたら使用を直ちに中止してください。そのまま使用すると、火災、故障の原因となります。 すぐにパソコンから抜き、煙などの異常が出なくなるのを確認し、販売店などに修理をご依頼ください。
分解禁止	▶分解・改造しないでください 分解、改造しないでください。怪我、感電、故障の原因となります。本製品の分解、改造による怪我や事故について、当社は責任を負いかねます。
接触禁止	▶濡れた手での操作は避けてください 濡れた手で電源ケーブル・プラグを抜き差ししないでください。また、製品に触れないでください。感電の原因となることがあります。

⚠注意

禁止	▶以下のような場所では使用しないでください 本製品を以下のような場所で使用すると、動作不良、故障の原因となります。 ・振動や衝撃が加わる場所 ・直射日光のあたる場所 ・湿気やホコリが多い場所 ・温度差の激しい場所 ・熱を発生するもの（暖房器具など）の近く ・強い磁力、電波が発生するもの（磁石、ディスプレイ、スピーカー、ラジオ、無線機など）の近く ・湿気の多い場所
子供注意	▶子供の手の届かない場所に置いてください 本製品に装着されている電子部品など子供が飲み込まないように注意してください。
引抜禁止	▶通信中はケーブルを引抜かないでください 本製品がパソコンと通信中の場合はケーブルを引抜かないでください。ケーブルの引抜きは、必ず通信していないときに行ってください。故障の原因になることがあります。
安全設計	▶安全設計をしてください 本製品を、高度な信頼性を必要とするシステムに使用する場合は、冗長設計、誤動作防止設計など充分な安全設計を必ず行ってください。本製品の故障、傷害により生じるいかなる損害、事故について当社は責任を負いかねます。
保管注意	▶長期間使用しない場合の保管について 長期間使用しない場合は、帯電防止袋などに入れ、ホコリなどが入らないようにしてください。ホコリや汚れが付着すると短絡、接触不良などの原因になります。
ホコリ注意	▶製品の清掃について 製品にホコリや汚れなどが付着すると短絡 、故障の原因になりますので 、下記の「▶お手入れについて」に従って清掃してください。
薬品注意	▶お手入れについて ホコリが付着した場合はサンハヤト製ジェットブロー（JBK-480）などのガススプレーで吹き飛ばしてください。
使用注意	▶故障、破損時の処理について 本製品が故障もしくは破損した場合は、速やかに使用を中止してください。そのまま使用しますと火災、感電、怪我の原因になるおそれがあります。
廃棄注意	▶本製品の廃棄について 本製品の廃棄は、各自治体の廃棄ルールに従ってください。詳しくは各自治体にお問い合わせください。

本資料についてのご注意

本資料について

・本資料は、電子工作や電子回路、パーソナルコンピュータの操作について一般的な知識をお持ちの方を対象にしています。
・本資料を元に操作するには、株式会社ルネサス エレクトロニクス製 H8/300H マイコンについての知識や開発環境などが必要です。
・Microsoft®、Windows® は米国 Microsoft 社の米国およびその他の国における登録商標です。
・その他、記載されている会社名、製品名は各社の商標または登録商標です。

本資料のご利用にあたって

・この取扱説明書に掲載している内容は、お客様が用途に応じた適切な製品をご購入頂くことを目的としています。その使用により当社及び第三者の知的財産権その他の権利に対する保証、又は実施権の許諾を意味するものではありません。また、権利の侵害に関して当社は責任を負いません。
・本資料に記載した情報を流用する場合は、お客様のシステム全体で充分評価し適用可能かご判断願います。当社では適用可能判断についての責任は負いません。
・本資料に記載してある内容は、一般的な電子機器（学習教材、事務機器、計測機器、パーソナル機器、コンピュータ機器など）に使用されることを目的としています。高い品質や信頼性が要求され、故障や誤作動が直接人命を脅かしたり人体に危害を及ぼす恐れのある、医療、軍事、航空宇宙、原子力制御、運輸、移動体、各種安全装置などの機器への使用は意図も保証もしておりません。
・この取扱説明書の一部、又は全部を著者および当社の承諾なしで、いかなる形でも転載又は複製されることは堅くお断りします。
・全ての情報は本資料発行時点のものであり、当社は予告なしに本資料に記載した内容を変更することがあります。
・この資料の内容は慎重に制作しておりますが、万一記述誤りによってお客様に損害が生じても当社はその責任を負いません。
・本資料に関してのお問合せ、その他お気付きの点がございましたら、当社までお問合せください。
・本資料に関する最新の情報はサンハヤト株式会社ホームページ（http://www.sunhayato.co.jp/）に掲載しております。

このテキストについて

本テキストでは、サンハヤト H8 マイコントレーニングボード MT-R300 のプログラム開発を、HEW（High-Performance Embedded Workshop）を使用して行った場合について説明しています。HEW の詳細な使用方法や注意事項につきましては、ルネサス エレクトロニクス社発行のマニュアルを参照してください。

サンハヤト H8 マイコントレーニングボード MT-R300 概要

● ターゲットマイコン ：H8/300H シリーズ 3062 グループ HD64H3062BF（以下 H8/3062BF マイコン）
● 書き込みボード ：MT-R300（H8/3062BF マイコン モード 5（内蔵 ROM 有効拡張 16M バイトモード）、モード 7（シングルチップアドバンストモード）対応）

開発に必要なもの

● 開発用 PC（以下の条件を満たすもの）

PC 本体	Pentium Ⅲ 600MHz 以上を搭載した IBM PC/AT 互換機
OS	Windows XP, Windows Me, Windows 98SE, Windows 2000（Windows XP 推奨）
メモリ	128MB 以上（ロードモジュールの 2 倍以上）
ハードディスク	エミュレータ用ソフトウェアのインストールには 100MB 以上の空き容量が必要
入力デバイス	マウス、またはマウス相当のポインティングデバイス
インターフェイス	1 ポート以上の USB インターフェイス

● サンハヤト H8 マイコントレーニングボード　MT-R300
● 付属の USB ケーブル

このマニュアルで使用しているツール類

このスタートアップガイドでは、以下の OS、ツールのバージョンで動作したものとして説明しています。

● ホスト PC の OS ：Windows XP Professional
● 統合化開発環境 ：HEW V.4.03.00001
● C コンパイラ ：H8S,H8/300 Standard Toolchain (V.6.02 Release00)
●フラッシュ書き込みツール ：FDT（FlashDevelopmentToolKit ）Ver4.02

HEW のバージョンは、HEW の「 ヘルプ 」メニューの「High-Performance Embedded Workshop のバージョン情報 」で確認できます。

このマニュアルで紹介している各社サイト、ツールについて

本スタートアップガイドで紹介している各社サイトやツールは、本スタートアップガイド発行時のものを掲載しております。各社サイトの構成やツールのバージョン、またバージョンに伴ってツールのファイル名などが変わっている場合がありますのでご了承ください。

目　次

1. はじめに

このボードは、グラフィック液晶ディスプレイ（Graphic Liquid Crystal Display）（以下「グラフィック LCD」という）モジュール（128 × 64 ドット（モノクローム））、モニター用 LED を設置した教育用ボードです。

このボードを用いて、グラフィック LCD モジュールのハードウエアの概要を習得します。続いて、そのインストラクション（命令（状態）、データ）を実行し、種々のグラフィック図形や自作の文字フォントを表示するプログラムを作成します。

このグラフィック LCD を制御するサンプルプログラムは、種々のテキスト、HP などに掲載されていますが、その多くは既存のライブラリーを活用しています。その詳細についてはあまり触れていません。ここでは、独自のライブラリーを作成し、理解を深めます。

モニター LED により、データバス線 8 本と制御信号線 6 本の信号をモニターできます。各信号の変化を視認することでより効果的な実習ができます。

2. 基礎知識

2.1 グラフィック LCD モジュール

(1) LCD パネル

このボードに搭載されているサンエンジニアリング社のグラフィック LCD モジュールについて説明します。その内容は、同社のデータシートから抜粋したもので、詳細はそちらを参照願います。

グラフィック LCD の基本原理等は、キャラクタ LCD とほぼ同じです。MT-E502 のテキストを参照願います。図 2-1 に LCD パネルのドット構成を示します。

カラム(Yアドレス)0〜63　カラム(Yアドレス)0〜63
チップ1　チップ2
ページ(Xアドレス)0〜7
縦:64ドット
横:128ドット
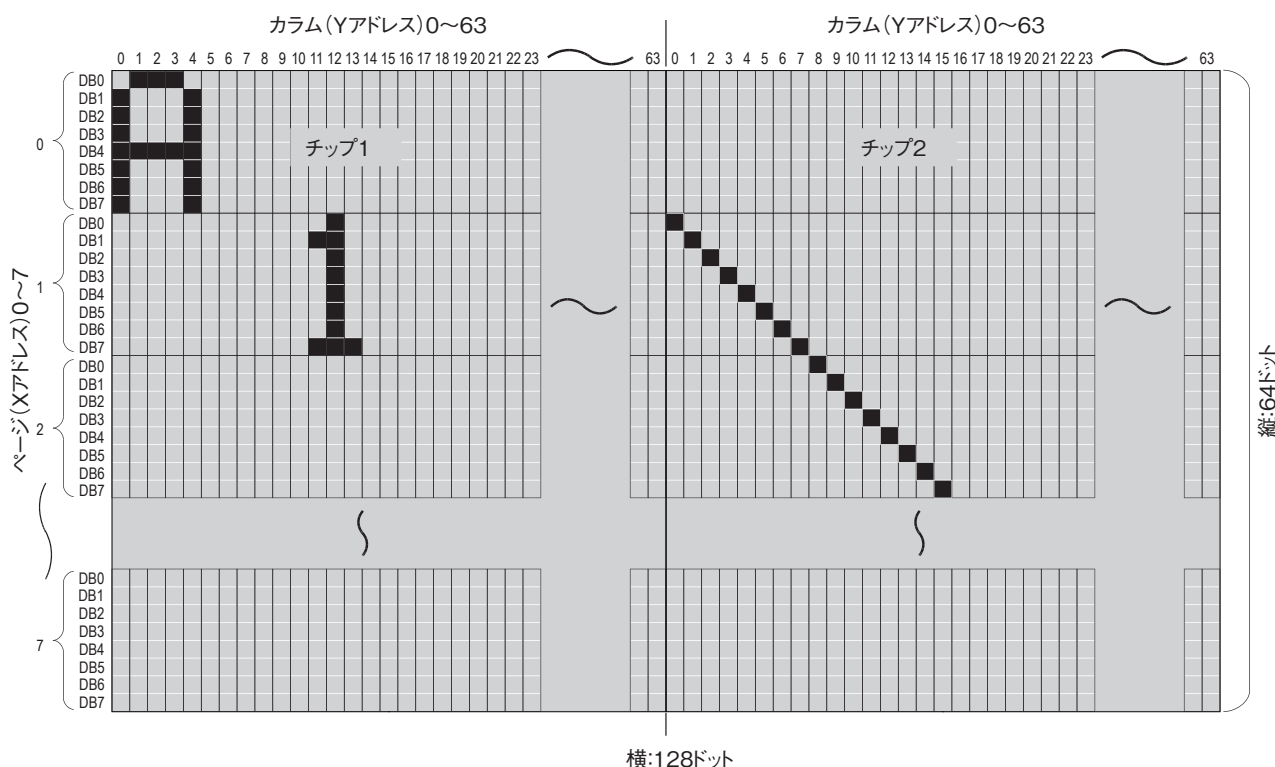
図 2-1　LCD パネルのドット構成

128 × 64 ドットで構成されていますが、実際には左右半分に分かれており、それぞれ 64 × 64 ドットの LCD チップで構成されています。左側をチップ 1、右側をチップ 2 と呼びます。1 つのチップ内では、縦方向をページ (0 〜 7)、横方向をカラム（0 〜 63）で指定し、DDRAM（データディスプレイ RAM）に表示データ（DB0 〜 7）を書き込むことで、パネルに 8 ドット単位で白 / 黒を表示します。逆に、ページ、カラムを指定し、DDRAM から表示データを読み出すこともできます。左右 2 つのチップで 128 × 64 のドットが制御できるため、ドットの集まりであるグラフィック図形やグラフなどが表示できます。

(2) LCD モジュールのブロック図

図 2-2 はそのブロック図を示します。このモジュールはマイコンからの動作許可信号（E)、データ / 命令選択信号（DI)、リード / ライト選択信号（RW)、リセット信号（RST)、チップ選択信号 2（CS2)、チップ選択信号 1（CS1）の 6 本の制御信号とデータバス信号（DB0 〜 7）により制御されます。モジュール内には共通ドライバー（KS0107）と 2 つのセグメントドライバー（KS0108）が設置されています。前者は CR 発振回路（OSC）を内蔵し、そこで生成されたクロック信号で横方向の電極（COM1 〜 64）を制御（選択）します。後者はデコーダーや DDRAM などを内蔵しています。デコーダーはインストラクションを解読し、適切に各部を制御します。データは DDRAM に読み書きします。さらにクロック信号と同期をとりながら DDRAM のデータを参照し、縦方向の電極（SEG1 〜 128）
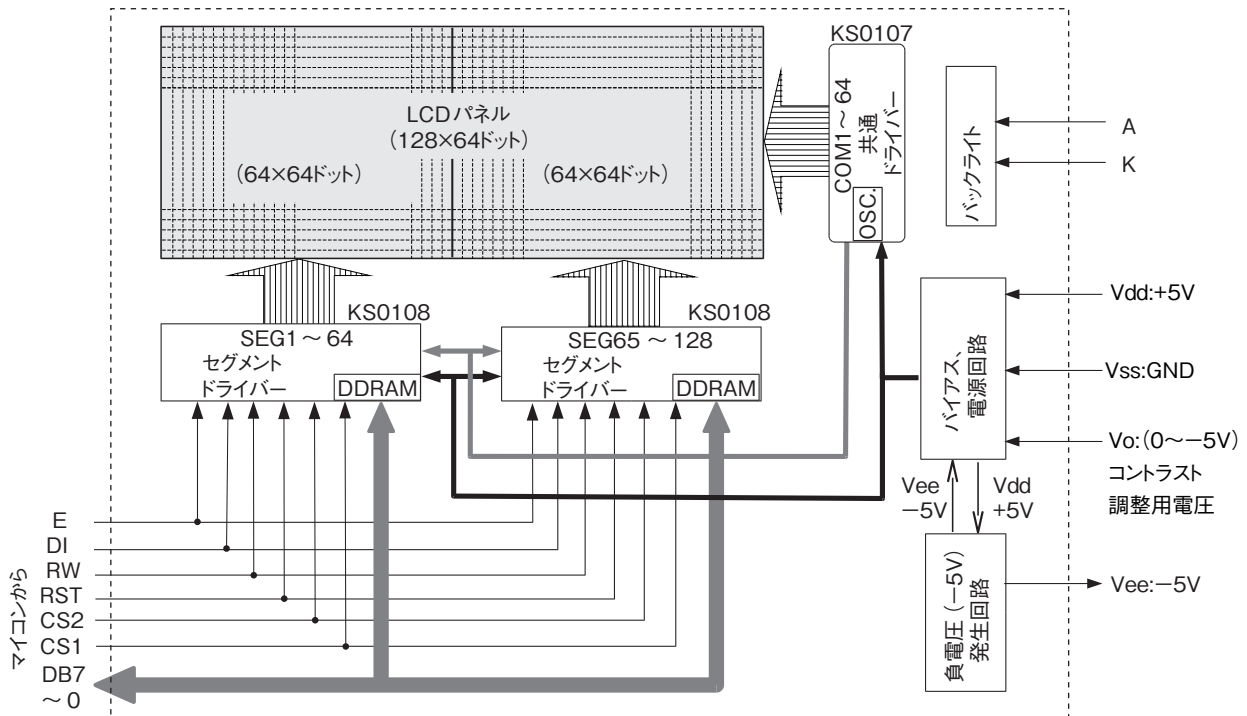
図 2-2　LCD モジュールのブロック図

を制御します。両者の協調により、LCD は動的に縦方向と横方向の電極の交点（ドット）の表示を制御しています。Vdd と Vss にはそれぞれ +5V、GND を供給します。これはロジック回路用電源となります。内部の負電圧発生回路では Vdd（+5V）から Vee（－ 5V）を発生させます。Vdd と Vee の電位差（10V）は LCD 用電源として使われます。外部に取り出された Vee はコントラスト調整用電圧 Vo を生成するために活用します。

グラフィック LCD モジュールは文字フォントが書き込まれた CGROM（Character Generator ROM）を組み込んでいないため、そのままでは文字を表示できません。

2.2　読み書きのタイミング

図 2-3 は LCD モジュールにインストラクション（命令（読み込み時は状態）、データ）を読み書きする時の各信号のタイミングチャートを示します。

（a）は書き込み時でその概要は次のとおりです。

① E（イネーブル）信号を L（動作禁止）に設定する。
② RW（リード / ライト）信号を L（書き込み）に設定する。
③ RST（リセット）信号を H（通常時）に設定する。
　DI（データ / インストラクション）信号を設定する。（データ時は H/ インストラクション時は L）
　CS1、CS2（チップ選択 1、2）信号を設定する。
　（チップ 1 選択時は CS2 信号 :H、CS1 信号 :L ／チップ 2 選択時は CS2 信号 :L、CS1 信号 :H）
④①から 450ns 以上、②から 140ns 以上待つ。
⑤ E（イネーブル）信号を H（動作許可）に設定する。
⑥ DB7 ～ 0（データ）にインストラクション（命令、データ）を書き込む。
⑦①から 1000ns 以上、⑤から 450ns 以上、⑥から 200ns 以上、待つ。
⑧ E（イネーブル）信号を L（動作禁止）に設定する。立下り時にインストラクションが受け渡される。

信号名(操作)
1000ns 以上(1サイクル)
450ns 以上　450ns 以上
E(1:許可／0:禁止)　0: 禁止　1: 許可　0: 禁止
140ns 以上　10ns 以上
RW(1:読込／0:書込)　0: 書込
140ns 以上　10ns 以上
RST(1:通常／0:リセット)
DI(1:データ／0:命令)
CS2(1:非選択／0:選択)
CS1(1:非選択／0:選択)　有　効
200ns 以上
DB0〜7　有　効

(a)書き込み

信号名(操作)
1000ns 以上(1サイクル)
450ns 以上　450ns 以上
E(1:許可／0:禁止)　0: 禁止　1: 許可　0: 禁止
140ns 以上　10ns 以上
RW(1:読込／0:書込)　1: 読出
140ns 以上　10ns 以上
RST(1:通常／0:リセット)
DI(1:データ／0:命令)
CS2(1:非選択／0:選択)
CS1(1:非選択／0:選択)　有　効
320ns 以下　20ns 以上
DB0〜7　有　効

(b)読み込み
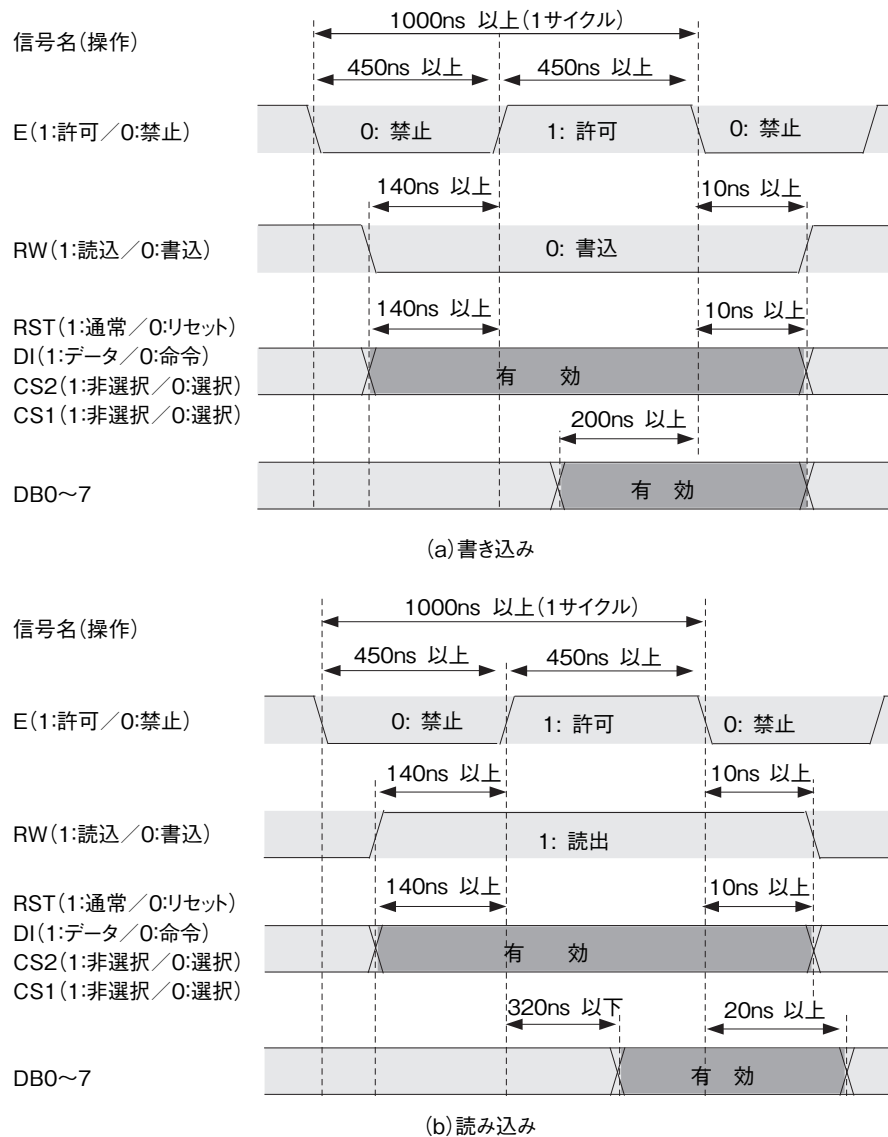
図 2-3　インストラクションの書き込み、読み出しのタイミングチャート

(b) は読み出し時でその概要は次のとおりです。

① E（イネーブル）信号を L（動作禁止）に設定する。
② RW（リード / ライト）信号を H（読み出し）に設定する。
③ RST（リセット）信号を H（通常時）に設定する。
　DI（データ / インストラクション）信号を設定する。(データ時は H/ インストラクション時は L)
　CS1、CS2（チップ選択 1、2）信号を設定する。
　(チップ 1 選択時は CS2 信号 :H、CS1 信号 :L ／チップ 2 選択時は CS2 信号 :L、CS1 信号 :H)
④①から 450ns 以上、②から 140ns 以上待つ。
⑤ E（イネーブル）信号を H（動作許可）に設定する。
⑥ DB7 〜 0（データ）からインストラクション（状態、データ）を読み出す。
⑦①から 1000ns 以上、⑤から 450ns 以上、待つ。
⑧ E（イネーブル）信号を L（動作禁止）に設定する。

CS2、CS1 信号は同時に L（両方の選択）にしないで下さい。故障の原因になります。

2.3 インストラクション

表 2-1 はインストラクションの一覧を示します。それぞれの使い方は各実習の中で解説します。

表 2-1　インストラクションの一覧

命令	D/I	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	機能
表示オン／オフ	0	0	0	0	1	1	1	1	1	表示	表示を制御する。 表示（0: オフ／ 1: オン） 内部ステータスやＤＤＲＡＭデータなどは変化しない。
カラムセット	0	0	0	1	カラム（0 ～ 63）						カラム (Ｙアドレス) カウンタにカラムをセットする。
ページセット	0	0	1	0	1	1	1	ページ（0 ～ 7）			ページ (Ｘアドレス) カウンタにページをセットする。
表示スタートライン	0	0	1	1	表示スタートライン（0 ～ 63）						画面最上位に表示されるＤＤＲＡＭアドレスをセットする。
ステータス読み出し	0	1	ビジー	0	表示	リセット	0	0	0	0	ステータス（状態）を読み込む ビジー（0: レディ／ 1: ビジー） 表示（0: オン／ 1: オフ） リセット(0:通常／ 1: リセット)
表示データ書き込み	1	0	表示データ								ＤＤＲＡＭにデータを書き込む 命令実行後、カラム (Ｙアドレス) カウンタが自動的にインクリメント (+1) される
表示データ読み込み	1	1	表示データ								ＤＤＲＡＭからデータを読み出す

2.4 初期設定

図 2-4 は LCD モジュールを動作させるための初期設定の手順で、その概要は次のとおりです。

①リセット信号を L レベルにした後、1 μ秒以上（ここでは 10 μ秒）待って、H レベルに戻す（リセット）。
②画面左半分（CS1）を選択（CS1=L､CS2=H）し､0x3f を書き込む（表示オン）。
③画面右半分（CS2）を選択（CS2=L､CS1=H）し､0x3f を書き込む（表示オン）。
④画面左半分（CS1）を選択（CS1=L､CS2=H）し､DDRAM の全アドレス（ページ =0 ～ 7、カラム =0 ～ 63）に 0 を書き込む（画面左半分クリア）。
⑤画面右半分（CS2）を選択（CS2=L､CS1=H）し､DDRAM の全アドレス（ページ =0 ～ 7、カラム =0 ～ 63）に 0 を書き込む（画面右半分クリア）。

初期設定
↓
リセット
（RST 信号を L（リセット）にする
1μ 秒以上（10μ 秒）待つ
RST 信号を H（通常）に戻す）
↓
画面左半分を選択し
表示をオンにする
↓
画面右半分を選択し
表示をオンにする
↓
画面左半分をクリア
（DDRAM の全アドレスに
0x00 を書き込む）
↓
画面右半分をクリア
（DDRAM の全アドレスに
0x00 を書き込む）

図 2-4　初期設定の手順

3. 接続図

図 3-1 は、このボードの接続図の概要を示します。

マイコンのポート VPD と VPE にそれぞれ制御信号線（E、DI、RW、RST、CS2、CS1）とデータバス線（DB0 ～ 7）を接続します。LCD モジュールの Vdd、Vss にはそれぞれ VPort のバスパワー（+5V、GND）を供給します。その内部で生成された Vee（－ 5V）と GND 間に可変抵抗器 VR を接続します。その摺動端子から可変電圧（0 ～－ 5V）を取り出し、Vo に接続します。Vo はコントラスト調整電源で、適宜調整します。このモジュールは LED バックライトがないタイプのため A（アノード）、K（カソード）は未接続とします。制御信号線、データバス線の H/L レベルをモニターするステータス LED が設置されています。

接続図の詳細は MT-E507 取扱説明書をご参照ください。
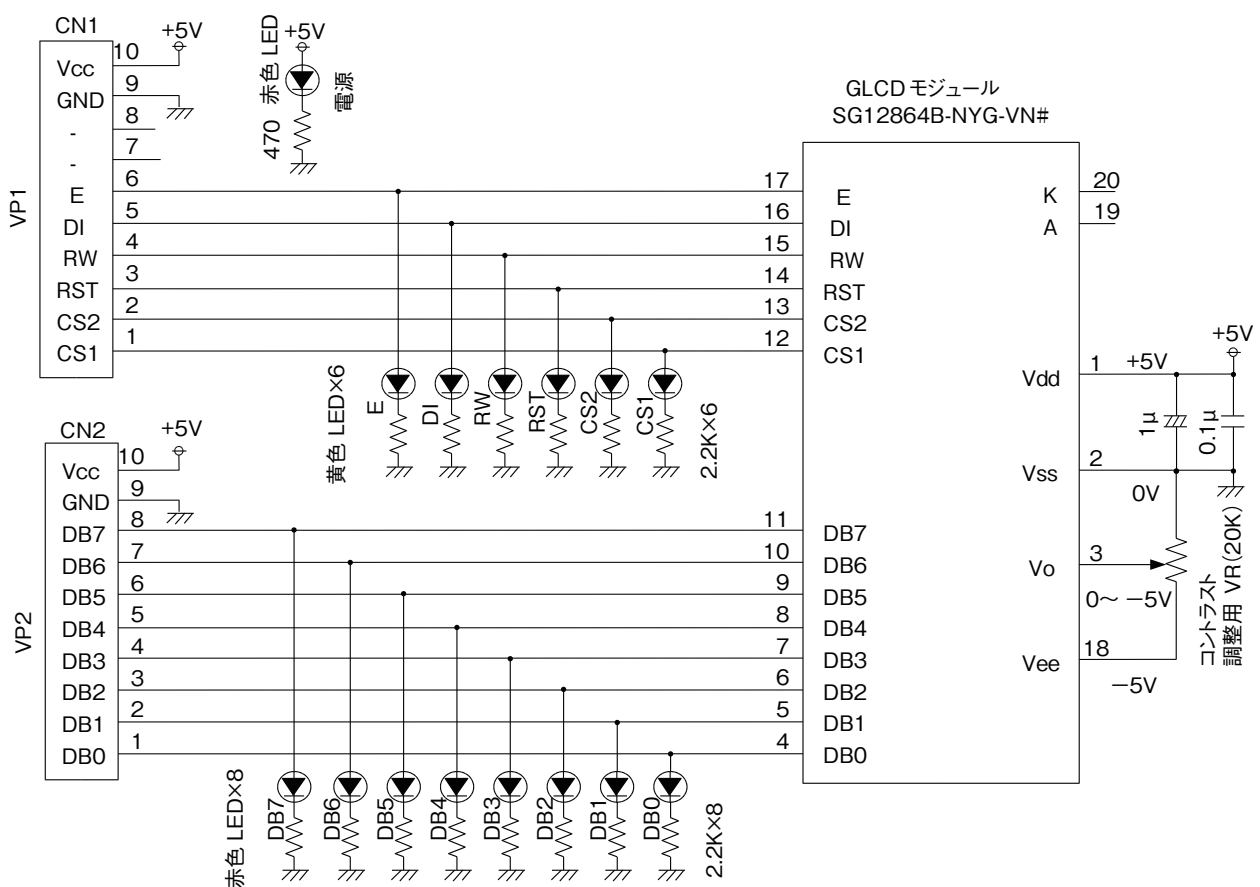
図 3-1　接続図の概要

4. 基本実習

ここではグラフィックLCDの制御方法を実習します。ドット、線分、円、文字フォントなどを表示する関数を作成し、それらをまとめたライブラリーを作成します。

4.1 基本プログラム

ここでは、グラフィック LCD モジュールを制御する種々の基本プログラムを作成します。

図 4-1 のとおり付属の 10P ケーブルを用いて、mt_R300 の VPD、VPE と mt_E507 のそれぞれ VP1、VP2 を接続します。

"X:\xxxx\workspace" にワークスペース、プロジェクト（glcd）を新規作成します。他の実習と同様に必要な設定をします。

図 4-1 基板間の接続

実習 4.1.1 グラフィック LCD に図 2-1 のドットデータを表示する

[解説]

図 4-2 は制御信号（VP1）のピン割り付けを示します。

これに基づき、#define 文を用いて 16 進数で各制御信号の状態を定義します。

- E 信号（bit5） ……………1: 動作許可（#define EN 0x20）／ 0: 動作禁止（#define DS 0x00）
- DI 信号（bit4）……………1: デ ー タ（#define DR 0x10）／ 0: 命令（状態）（#define IR 0x00）
- RW 信号（bit3）……………1: 読み出し（#define RD 0x08）／ 0: 書き込み（#define WT 0x00）
- RST 信号（bit2） …………1: 通常（#define NML 0x04）／ 0: リセット（#define RST 0x00）
- CS2、1 信号（bit1、0）… 1: 非選択／ 0: 選択（CS2、CS1 の同時選択は禁止）
  右チップ（CS2）の選択（CS2=0、CS1=1）（#define CS2 0x01）
  左チップ（CS1）の選択（CS1=0、CS2=1）（#define CS1 0x02）

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
NC (入力)	NC (入力)	E (出力)	DI (出力)	RW (出力)	RST (出力)	CS2 (出力)	CS1 (出力)
		イネーブル信号	データ／インストラクション選択信号	リード／ライト選択信号	リセット信号	画面右半分選択信号	画面左半分選択信号
未接続	未接続	1:動作許可 0:動作禁止	1:データ 0:命令(状態)	1:読み出し 0:書き込み	1:通常 0:リセット	1:非選択 0:選択	1:非選択 0:選択

図 4-2 制御信号のピン割り付け
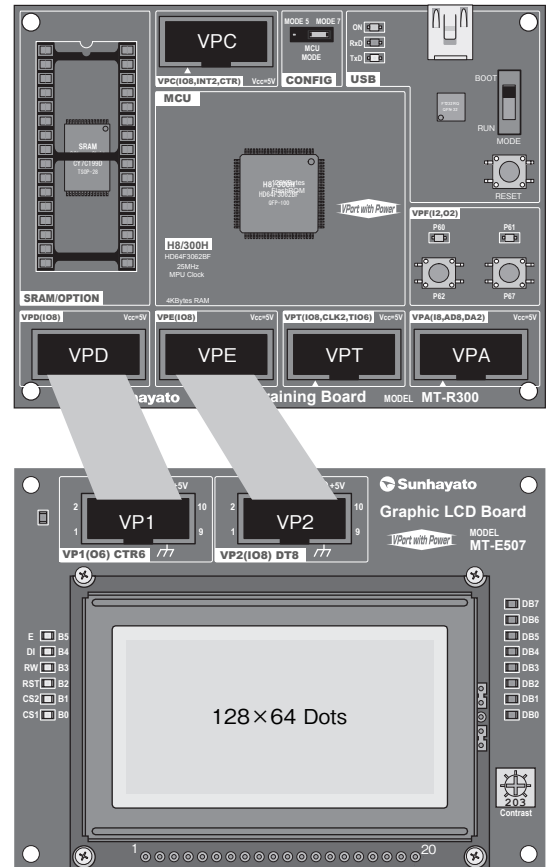
図 2-3（a）のタイミングチャートに従い、グラフィック LCD にインストラクションを書き込む関数を作成します。

実際には命令とデータに分けて次の 2 つの関数を作成します。

```
void write_cmnd_glcd( int cs, char cmnd);        // 命令を書き込む関数
void write_data_glcd( int cs, char data);        // データを書き込む関数
```

cs には CS1 または CS2 を代入します。
これらの関数を用いてメインプログラムを作成します。
グラフィック LCD は次のように初期化します（表 2-1 のインストラクション一覧表参照)。

・RST 端子を一定時間（10 μ秒）L レベルにし、H レベルに戻す。（リセット動作）
　仕様では 1 μ秒以上ですが、5 〜 10 μ秒は必要です。
・画面左半分の表示をオンにする。（CS1 に '00111111' を書き込む。）
・画面右半分の表示をオンにする。（CS2 に '00111111' を書き込む。）
・画面左半分の表示をクリアする。(ページ（X アドレス）：（0 〜 7)、カラム（Y アドレス）：（0 〜 63）
　を順次セットし、CS1 のディスプレイデータ RAM 全体に 0x00（空白）を書き込む。）
・画面右半分の表示をクリアする。（ページ（X アドレス）：（0 〜 7)、カラム（Y アドレス）：（0 〜 63）
　を順次セットし、CS2 のディスプレイデータ RAM 全体に 0x00（空白）を書き込む。）

次に、図 2-1（'A','1'、斜線）のような表示をします。
'A' は次のように表示します。

・画面左半分（CS1）に対し、ページ（X アドレス）に 0 をセットし、カラム (Y アドレス) に初期値 0 をセットします。
・DDRAM に順次以下のデータ（'A' のフォント）を書き込む。カラム (Y アドレス) は自動的にインクリメントされるため、その設定は省略します。

```
0xfe    '■■■■■■■□'    2 進数では '11111110'
0x11    '□□□■□□□■'    2 進数では '00010001'
0x11    '□□□■□□□■'    2 進数では '00010001'
0x11    '□□□■□□□■'    2 進数では '00010001'
0xfe    '■■■■■■■□'    2 進数では '11111110'
```

次の '1' と斜線はご自身で考えて下さい。

[解答]

リスト 4.1.1 にプログラム例を示します。

リスト 4.1.1　グラフィック LCD に図 1 のドットデータを表示する

```
/** main.c  ****************************************************************************/
#include    <mt_r300.h>                     // マイコンボード用ライブラリー
//  制御信号 (b7:NC, b6:NC, b5:E ,b4:DI ,b3:RW ,b2:RST ,b1:CS2 ,b0:CS1)
#define EN       0x20                       // E=1  ( 動作許可 )
#define DS       0x00                       // E=0  ( 動作禁止 )
#define DR       0x10                       // DI=1 ( データレジスタ )
#define IR       0x00                       // DI=0 ( 命令レジスタ )
#define RD       0x08                       // RW=1 ( 読み出し )
```

```
#define WT       0x00                          // RW=0 （書き込み）
#define  NML     0x04                          // RST=1(通常）
#define  RST     0x00                          // RST=0(リセット）
#define  CS2     0x01                          // 右チップ(CS2)選択(CS2=0,CS1=1)
#define  CS1     0x02                          // 左チップ(CS1)選択(CS1=0,CS2=1)

void write_cmnd_glcd( int cs, char cmnd)   // 命令を書き込む関数
{
    wait_us_itu2(40);                      // ビジー時間待つ
    out_VPD( NML | cs | IR | WT | DS );    // DS:動作禁止，他制御信号設定
                                           // Twl(450ns)以上待つ(VPort関数の実行時間により不要）
    out_VPD( NML | cs | IR | WT | EN );    // EN:動作許可
    out_VPE(cmnd);                         // 命令を設定
                                           // Tdsu(200ns)以上，動作許可後Twh(450ns)以上待つ(不要）
    out_VPD( NML | cs | IR | WT | DS );    // DS:動作禁止(EN-->DSの立下りで書き込む）
}
void write_data_glcd( int cs, char data)   // データを書き込む関数
{
    wait_us_itu2(40);                      // ビジー時間待つ
    out_VPD( NML | cs | DR | WT | DS );    // DS:動作禁止，他制御信号設定

                                           // Twl(450ns)以上待つ(VPort関数の実行時間により不要）
    out_VPD( NML | cs | DR | WT | EN );    // EN:動作許可
    out_VPE(data);                         // データを設定

                                           // Tdsu(200ns)以上，動作許可後Twh(450ns)以上待つ(不要）
    out_VPD( NML | cs | DR | WT | DS );    // DS:動作禁止(EN-->DSの立下りで書き込む）
}
void main(void)
{
    int page, clmn;
    setup_VPD(0x3f);                       // 初期設定(b5:E,b4:DI,b3:RW,b2:RST,b1:CS2,b0:CS1)
    setup_VPE(0xff);                       // 初期設定(インストラクション）
    out_VPD( RST );                        // リセット(RST=L)
    wait_us_itu2(10);                      // 10μs待つ(仕様:1μs以上）
    out_VPD( NML );                        // 通常(RST=H)
    write_cmnd_glcd( CS1, 0x3f);           // CS1の表示オン
    write_cmnd_glcd( CS2, 0x3f);           // CS2の表示オン
    // データのクリア
    for (page = 0; page < 8; page++) {             // ページ(0～7)
        write_cmnd_glcd( CS1, 0xb8 | page );       // ページセット(0:上～7:下）
        write_cmnd_glcd( CS1, 0x40 | 0 );          // カラムに初期値0をセット
        for (clmn = 0; clmn < 64; clmn++) {        // カラム(0～63)
//          write_cmnd_glcd( CS1, 0x40 | clmn);    // カラムセット(自動インクリにより省略）
            write_data_glcd(CS1, 0x00);            // データ(空白）の書き込み
```

```
        }
    }
    for (page = 0; page < 8; page++) {             // ページ(0～7)
        write_cmnd_glcd( CS2, 0xb8 | page );       // ページセット(0:上～7:下)
        write_cmnd_glcd( CS1, 0x40 | 0 );          // カラムに初期値0をセット
        for (clmn = 0; clmn < 64; clmn++) {        // カラム(0～63)
//          write_cmnd_glcd( CS2, 0x40 | clmn);    // カラムセット(自動インクリにより省略)
            write_data_glcd(CS2, 0x00);            // データ(空白)の書き込み
        }
    }
    wait_ms(500);
    // データの書き込み
    write_cmnd_glcd( CS1, 0xb8 | 0 );              // ページセット
    write_cmnd_glcd( CS1, 0x40 | 0 );              // カラムセット
    write_data_glcd( CS1, 0xfe );                  // 'A'  ■■■■■■■□
    write_data_glcd( CS1, 0x11 );                  //      □□□■□□□■
    write_data_glcd( CS1, 0x11 );                  //      □□□■□□□■
    write_data_glcd( CS1, 0x11 );                  //      □□□■□□□■
    write_data_glcd( CS1, 0xfe );                  //      ■■■■■■■□
    write_cmnd_glcd( CS1, 0xb8 | 1 );              // ページセット
    write_cmnd_glcd( CS1, 0x40 | 10 );             // カラムセット
    write_data_glcd( CS1, 0x00 );                  // '1'  □□□□□□□□
    write_data_glcd( CS1, 0x82 );                  //      ■□□□□□■□
    write_data_glcd( CS1, 0xff );                  //      ■■■■■■■■
    write_data_glcd( CS1, 0x80 );                  //      ■□□□□□□□
    write_data_glcd( CS1, 0x00 );                  //      □□□□□□□□
    write_cmnd_glcd( CS2, 0xb8 | 1 );              // ページセット
    write_cmnd_glcd( CS2, 0x40 | 0 );              // カラムセット
    write_data_glcd( CS2, 0x01 );                  // '線' □□□□□□□■
    write_data_glcd( CS2, 0x02 );                  //      □□□□□□■□
    write_data_glcd( CS2, 0x04 );                  //      □□□□□■□□
    write_data_glcd( CS2, 0x08 );                  //      □□□□■□□□
    write_data_glcd( CS2, 0x10 );                  //      □□□■□□□□
    write_data_glcd( CS2, 0x20 );                  //      □□■□□□□□
    write_data_glcd( CS2, 0x40 );                  //      □■□□□□□□
    write_data_glcd( CS2, 0x80 );                  //      ■□□□□□□□
    write_cmnd_glcd( CS2, 0xb8 | 2 );              // ページセット
    write_cmnd_glcd( CS2, 0x40 | 8 );              // カラムセット
    write_data_glcd( CS2, 0x01 );                  // '線' □□□□□□□■
    write_data_glcd( CS2, 0x02 );                  //      □□□□□□■□
    write_data_glcd( CS2, 0x04 );                  //      □□□□□■□□
    write_data_glcd( CS2, 0x08 );                  //      □□□□■□□□
    write_data_glcd( CS2, 0x10 );                  //      □□□■□□□□
    write_data_glcd( CS2, 0x20 );                  //      □□■□□□□□
    write_data_glcd( CS2, 0x40 );                  //      □■□□□□□□
```

```
    write_data_glcd( CS2, 0x80 );                 //      ■□□□□□□□
    while (1);
}
```

実習 4.1.2 グラフィック LCD に図 2-1 のドットデータを表示する（ビジーチェック付き）

[解説]

図 2-3（b）のタイミングチャートに従い、グラフィック LCD からインストラクションを読み出す関数を作成します。実際には、ステータス（状態）とデータに分けて次の 2 つの関数を作成します。

```
unsigned char read_stat_glcd( int cs )   // 状態を読み出す関数
unsigned char read_data_glcd( int cs )   // データを読み出す関数
```

cs には CS1 または CS2 を代入します。

状態を読み出し、BF（ビット 7）でグラフィック LCD がビジーかレディーかが判ります。ビジーチェックを繰り返すことで、直前のインストラクションの内部処理が終わり次第、次のインストラクションを実行します。ムダなウエイトがなくなり処理速度がアップします。

[解答]

リスト 4.1.2 にプログラム例を示します。

リスト 4.1.2　グラフィック LCD に図 1 のドットデータを表示する（ビジーチェック付き）

```
/** main.c  ******************************************************************************/
#include    <mt_r300.h>                   // マイコンボード用ライブラリー

この間の #define 文（10 行）は、実習 4.1.1 と全く同じためそれをコピー・ペーストして下さい。

unsigned char read_stat_glcd( int cs )    // ステータス（状態）を読み出す関数
{
    char stat;
    setup_VPE(0x00);                      // 一時的に入力モードに設定
    out_VPD( NML | cs | IR | RD | DS );   // DS: 動作禁止，他制御信号設定
    out_VPD( NML | cs | IR | RD | EN );   // EN: 動作許可
    stat = in_VPE();                      // 状態を読み出す
    out_VPD( NML | cs | IR | RD | DS );   // DS: 動作禁止
    setup_VPE(0xff);                      // 本来の出力モードに設定
    return stat;
}
unsigned char read_data_glcd( int cs )    // データを読み出す関数
{
    char data;
    while(read_stat_glcd(cs) & 0x80);     // ビジー (BF=1(0x80)) のとき繰り返す ( 待つ )
    setup_VPE(0x00);                      // 一時的に入力モードに設定
    out_VPD( NML | cs | DR | RD | DS );   // DS: 動作禁止，他制御信号設定
    out_VPD( NML | cs | DR | RD | EN );   // EN: 動作許可
    data = in_VPE();                      // データを読み出す
```

```
    out_VPD( NML | cs | DR | RD | DS );    // DS:動作禁止
    setup_VPE(0xff);                       // 本来の出力モードに設定
    return data;
}
void write_cmnd_glcd( int cs, char cmnd)  // 命令を書き込む関数
{
    while(read_stat_glcd(cs) & 0x80);      // ビジー(BF=1(0x80))のとき繰り返す(待つ)
    out_VPD( NML | cs | IR | WT | DS );    // DS:動作禁止,他制御信号設定
    out_VPD( NML | cs | IR | WT | EN );    // EN:動作許可
    out_VPE(cmnd);                         // 命令を設定
    out_VPD( NML | cs | IR | WT | DS );    // DS:動作禁止(EN-->DSの立下りで書き込む)
}
void write_data_glcd( int cs, char data)  // データを書き込む関数
{
    while(read_stat_glcd(cs) & 0x80);      // ビジー(BF=1(0x80))のとき繰り返す(待つ)
    out_VPD( NML | cs | DR | WT | DS );    // DS:動作禁止,他制御信号設定
    out_VPD( NML | cs | DR | WT | EN );    // EN:動作許可
    out_VPE(data);                         // データを設定
    out_VPD( NML | cs | DR | WT | DS );    // DS:動作禁止(EN-->DSの立下りで書き込む)
}
void main(void)
{
    main()は、実習4.1.1と全く同じためそれをコピー・ペーストして下さい。
}
```

4.2　グラフィックLCD制御関数のライブラリー作成

ここでは4.1で作成したグラフィックLCD制御関数をライブラリーにまとめます。

"X:\xxxx\workspace"にワークスペース、プロジェクト(glcd_lib)を新規作成し、4.1と同様に種々の設定をします。"mt_e507.c"、"mt_e507.h"を新規作成し(いずれも空白)、"X:\xxxx\workspace\@library"に保存します。"mt_r300.c"と同様に"mt_e507.c"もプロジェクトに追加します。

実習4.2.1　グラフィックLCDに図2-1のドットデータを書き込む

[解説]

ライブラリー"mt_e507.c"に実習4.1.2のグラフィックLCD制御関数を移植し、その他必要な関数を適宜作成します。合わせて"mt_e507.h"にその関数のプロトタイプ宣言を作成します。

[解答]

リスト4.2.1にプログラム例を示します。

リスト4.2.1　グラフィックLCDに図1のドットデータを書き込む

```
/** main.c  ***************************************************************************/
#include    <mt_r300.h>                   // マイコンボード用ライブラリー
#define  CS2     0x01                     // 右チップ(CS2)選択(CS2=0,CS1=1)
#define  CS1     0x02                     // 左チップ(CS1)選択(CS1=0,CS2=1)
```

```
void main(void)
{
    initialize_glcd();                                                          // GLCD の初期化
    wait_ms(500);
    write_cmnd_glcd( CS1, 0xb8 | 0 );       // ページセット
    write_cmnd_glcd( CS1, 0x40 | 0 );       // カラムセット
    write_data_glcd( CS1, 0xfe );           // 'A'  ■■■■■■■□
    write_data_glcd( CS1, 0x11 );           //      □□□■□□□■
    write_data_glcd( CS1, 0x11 );           //      □□□■□□□■
    write_data_glcd( CS1, 0x11 );           //      □□□■□□□■
    write_data_glcd( CS1, 0xfe );           //      ■■■■■■■□
    write_cmnd_glcd( CS1, 0xb8 | 1 );       // ページセット
    write_cmnd_glcd( CS1, 0x40 | 10 );      // カラムセット
    write_data_glcd( CS1, 0x00 );           // '1'  □□□□□□□□
    write_data_glcd( CS1, 0x82 );           //      ■□□□□□■□
    write_data_glcd( CS1, 0xff );           //      ■■■■■■■■
    write_data_glcd( CS1, 0x80 );           //      ■□□□□□□□
    write_data_glcd( CS1, 0x00 );           //      □□□□□□□□
    write_cmnd_glcd( CS2, 0xb8 | 1 );       // ページセット
    write_cmnd_glcd( CS2, 0x40 | 0 );       // カラムセット
    write_data_glcd( CS2, 0x01 );           // '線' □□□□□□□■
    write_data_glcd( CS2, 0x02 );           //      □□□□□□■□
    write_data_glcd( CS2, 0x04 );           //      □□□□□■□□
    write_data_glcd( CS2, 0x08 );           //      □□□□■□□□
    write_data_glcd( CS2, 0x10 );           //      □□□■□□□□
    write_data_glcd( CS2, 0x20 );           //      □□■□□□□□
    write_data_glcd( CS2, 0x40 );           //      □■□□□□□□
    write_data_glcd( CS2, 0x80 );           //      ■□□□□□□□
    write_cmnd_glcd( CS2, 0xb8 | 2 );       // ページセット
    write_cmnd_glcd( CS2, 0x40 | 8 );       // カラムセット
    write_data_glcd( CS2, 0x01 );           // '線' □□□□□□□■
    write_data_glcd( CS2, 0x02 );           //      □□□□□□■□
    write_data_glcd( CS2, 0x04 );           //      □□□□□■□□
    write_data_glcd( CS2, 0x08 );           //      □□□□■□□□
    write_data_glcd( CS2, 0x10 );           //      □□□■□□□□
    write_data_glcd( CS2, 0x20 );           //      □□■□□□□□
    write_data_glcd( CS2, 0x40 );           //      □■□□□□□□
    write_data_glcd( CS2, 0x80 );           //      ■□□□□□□□
    while (1);
}
```

```
/** mt_e507.c ( 新規作成 ) ***********************************************************************/
#include    <mt_r300.h>                     // マイコンボード用ライブラリー
//  制御信号(b7:NC, b6:NC, b5:E ,b4:DI ,b3:RW ,b2:RST ,b1:CS2 ,b0:CS1)
#define EN       0x20                       // E=1  ( 動作許可 )
```

```
#define DS       0x00                        // E=0  （動作禁止）
#define DR       0x10                        // DI=1 （データレジスタ）
#define IR       0x00                        // DI=0 （命令レジスタ）
#define RD       0x08                        // RW=1 （読み出し）
#define WT       0x00                        // RW=0 （書き込み）
#define  NML     0x04                        // RST=1（通常）
#define  RST     0x00                        // RST=0（リセット）
#define  CS2     0x01                        // 右チップ（CS2）選択（CS2=0,CS1=1）
#define  CS1     0x02                        // 左チップ（CS1）選択（CS1=0,CS2=1）

char read_stat_glcd( int cs )                // ステータス（状態）を読み出す関数（cs:チップセレクト）
{
    char stat;
    setup_VPE(0x00);                         // VPE を一時的に入力モードに設定
    out_VPD( NML | cs | IR | RD | DS );      // DS:動作禁止，他制御信号設定
    out_VPD( NML | cs | IR | RD | EN );      // EN:動作許可
    stat = in_VPE();                         // 状態を読み出す
    out_VPD( NML | cs | IR | RD | DS );      // DS:動作禁止
    setup_VPE(0xff);                         // VPE を本来の出力モードに設定
    return stat;
}
/* void busy_check_glcd( int cs )            // ビジーチェック関数（cs:チップセレクト）
{
    while(read_stat_glcd( cs ) & 0x80);      // ビジー（BF=1(0x80)）のとき繰り返す（待つ）
} */
char read_data_glcd( int cs )                //データを読み出す関数（cs:チップセレクト）
{
    char data;
//  busy_check_glcd( cs );                   // ビジーチェック関数（次行で代行）
    while(read_stat_glcd( cs ) & 0x80);      // ビジー（BF=1(0x80)）のとき繰り返す（待つ）
    setup_VPE(0x00);                         // VPE を一時的に入力モードに設定
    out_VPD( NML | cs | DR | RD | DS );      // DS:動作禁止，他制御信号設定
    out_VPD( NML | cs | DR | RD | EN );      // EN:動作許可
    data = in_VPE();                         // データを読み出す
    out_VPD( NML | cs | DR | RD | DS );      // DS:動作禁止
    setup_VPE(0xff);                         // VPE を本来の出力モードに設定
    return data;                             // 戻り値：データ
}
void write_cmnd_glcd( int cs, char cmnd)     //命令を書き込む関数（cs:チップセレクト，cmnd:命令）
{
//  busy_check_glcd( cs );                   // ビジーチェック関数（次行で代行）
    while(read_stat_glcd( cs ) & 0x80);      // ビジー（BF=1(0x80)）のとき繰り返す（待つ）
    out_VPD( NML | cs | IR | WT | DS );      // DS:動作禁止，他制御信号設定
    out_VPD( NML | cs | IR | WT | EN );      // EN:動作許可
    out_VPE(cmnd);                           // 命令を設定
```

```
    out_VPD( NML | cs | IR | WT | DS );    // DS:動作禁止(EN-->DSの立下りで書き込む)
}
void write_data_glcd( int cs, char data)  // データを書き込む関数(cs:チップセレクト,data:データ)
{
//  busy_check_glcd( cs );                 // ビジーチェック関数(次行で代行)
    while(read_stat_glcd( cs ) & 0x80);    // ビジー(BF=1(0x80))のとき繰り返す(待つ)
    out_VPD( NML | cs | DR | WT | DS );    // DS:動作禁止,他制御信号設定
    out_VPD( NML | cs | DR | WT | EN );    // EN:動作許可
    out_VPE(data);                         // データを設定
    out_VPD( NML | cs | DR | WT | DS );    // DS:動作禁止(EN-->DSの立下りで書き込む)
}
void display_sw_glcd( int sw )             // 表示を制御する関数(sw:表示スイッチ(1:オン/0:オフ))
{
    write_cmnd_glcd( CS1, 0x3e | ( sw & 0x01 ) );          // CS1の表示制御
    write_cmnd_glcd( CS2, 0x3e | ( sw & 0x01 ) );          // CS2の表示制御
}
void set_page( int cs, int page)           // ページを設定する関数(cs:チップセレクト,page:ページ)
{
    write_cmnd_glcd( cs, 0xb8 | ( page & 0x07 ));          // ページ(上0～7下)
}
void set_clmn( int cs, int clmn)           // カラムを設定する関数(cs:チップセレクト,clmn:カラム)
{
    write_cmnd_glcd( cs, 0x40 | ( clmn & 0x3f ));          // カラム(左0～63右)
}
void set_sline( int cs, int sline) // 表示開始ラインを設定する関数(cs:チップセレクト,sline:開始ライン)
{
    write_cmnd_glcd( cs, 0xc0 | ( sline & 0x3f ));         // 表示開始ライン(0～63)
}
void clear_glcd(void)                                     // 表示をクリアする関数
{
    int page, clmn;
    for (page = 0; page < 8; page++) {                    // ページ(0～7)
        set_page(CS1, page);                              // CS1のページセット
        set_page(CS2, page);                              // CS2のページセット
        set_clmn(CS1, 0);                                 // CS1のカラムセット(初期値:0)
        set_clmn(CS2, 0);                                 // CS2のカラムセット(初期値:0)
        for (clmn = 0; clmn < 64; clmn++) {               // カラム(0～63)は自動インクリメント
            write_data_glcd(CS1, 0x00);                   // CS1のディスプレイデータのクリア
            write_data_glcd(CS2, 0x00);                   // CS2のディスプレイデータのクリア
        }
    }
}
void reset_glcd(void)                      // リセットする関数
{
    out_VPD( RST );                        // リセット(RST=L)
```

```
    wait_us_itu2(10);                // 10μs待つ(仕様:1μs以上)
    out_VPD( NML );                  // 通常(RST=H)
}
void initialize_glcd(void)           // 初期化する関数
{
    setup_VPD(0x3f);                 // ポートの設定(b5:E,b4:DI,b3:RW,b2:RST,b1:CS2,b0:CS1)
    setup_VPE(0xff);                 // ポートの設定(インストラクション)
    reset_glcd();                    // リセット
    display_sw_glcd(1);              // 表示オン(デフォルト値:オフ)
    clear_glcd();                    // 表示クリア
}
```

```
/** mt_e507.h (新規作成) *****************************************************************/
char    read_stat_glcd( int );
//void  busy_check_glcd( int );
char    read_data_glcd( int );
void    write_cmnd_glcd( int , char );
void    write_data_glcd( int , char );
void    display_sw_glcd( int );
void    set_page( int , int );
void    set_clmn( int , int );
void    set_sline( int , int );
void    clear_glcd( void );
void    reset_glcd( void );
void    initialize_glcd( void );
```

表示開始ラインを設定する関数は、ここでは使用していませんが参考に作成しました。

実習 4.2.2a グラフィック LCD の座標（67,51）に 1 ドット（黒）を描く（図 4-3 参照）

[解説]

次のように、与えられた x,y 座標からチップセレクト、ページ、カラム、ドットデータ等を算出します。

・（x,y）座標が有効範囲（x:0 ～ 127, y:0 ～ 63）かをチェックする（リミットチェック）。
・y 座標の 0 は下側、page の 0 は上側のため、y 座標を上下反転し、y = 63 – y; とします。
・チップセレクトは x ＜ 64 の時 CS1（=2）、x ＜＝ 64 の時 CS2（=1）のため、cs = 2 - x / 64; で求めます。
・カラムは x を 64 で割った余りで、clmn = x % 64; で求めます。
・ページは x を 64 で割った商で、page = y / 64; で求めます。
・（y % 8）はドットデータの何ビット目かを示すため、ドットデータは 0x01 を（y % 8）ビット左にシフトして求めます。

[解答]

リスト 4.2.2a にプログラム例を示します。
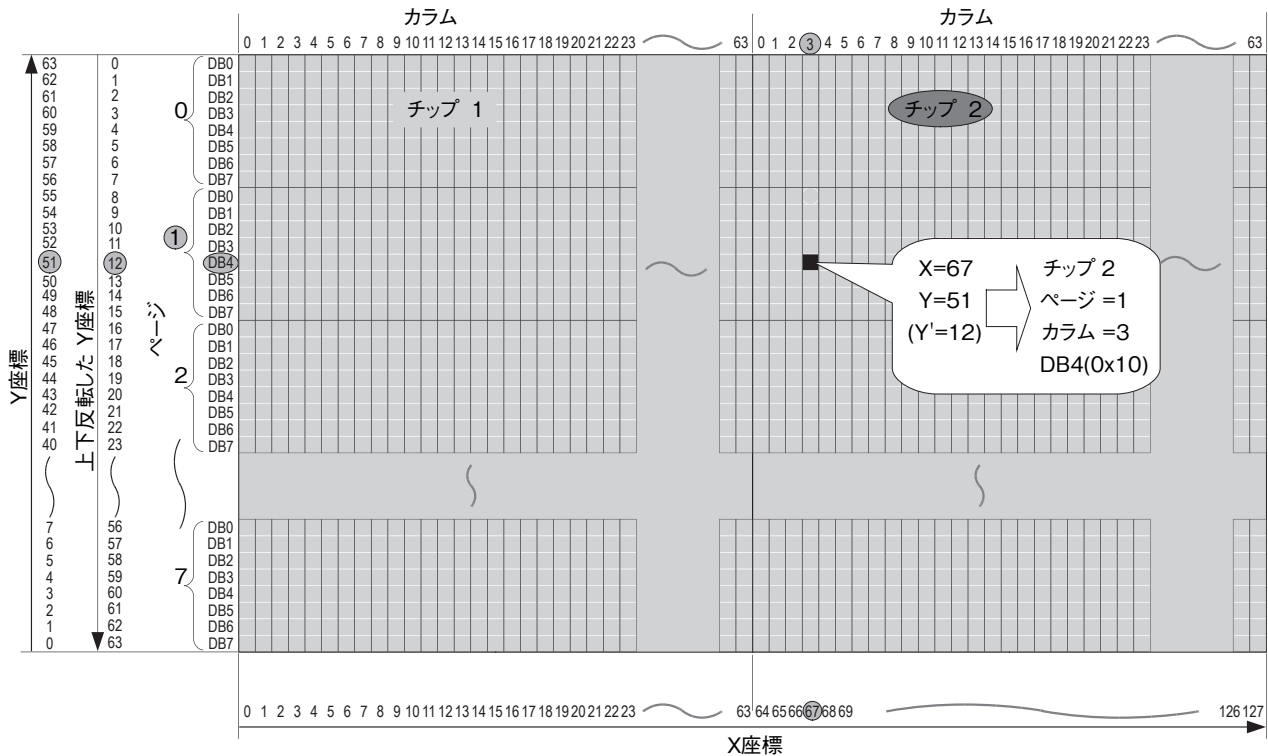
図 4-3　グラフィック LCD の座標（67,51）に 1 ドット（黒）を描く

リスト 4.2.2a　グラフィック LCD の座標（67,51）に 1 ドット（黒）を描く（図 4-3 参照）

```
/** main.c  ******************************************************************************/
#include    <mt_r300.h>                                     // マイコンボード用ライブラリー
#include    <mt_e507.h>                                     // グラフィック LCD ボード用ライブラリー
void main(void)
{
    int x, y, col, cs, clmn, page, bit;
    char data;
    initialize_glcd();                                      // GLCD の初期化
    wait_ms(500);
    x = 67, y = 51, col = 1;                                // xy 座標 (67,51), 色 ( 黒 (1))
    if( (0 <= x && x < 128) && (0 <= y && y < 64) )         // 有効座標？ ((0,0) 〜 (+127,+63))
    {
        cs = 2 - x / 64;                                    // チップセレクト (cs=2-67/64=2-1=1=CS2)
        clmn = x % 64;                                      // カラム (clmn=3)
        page = (63 - y) / 8;                                // ページ (page=(63-51)/8=12/8=1)
        bit = (63 - y) % 8;                                 // ビット (bit=(63-51)%8=12%8=4)
        set_page( cs, page );                               // ページセット (1)
        set_clmn( cs, clmn );                               // カラムセット (3)
        read_data_glcd( cs );                               // DDRAM データ ( ダミー ) を読み出す
        data = read_data_glcd( cs );                        // DDRAM データ ( 有効 ) を読み出す
        set_clmn( cs, clmn );                               // カラム再セット ( 自動インクリメント )
        if( col )                                           // カラー？ (1: 黒 /0: 白 )
            write_data_glcd( cs, data | (1 << bit) );       // 読出したデータの 1 ドットを黒く
        else                                                // 1<<bit = 1<<4 = 0x10
```

```
            write_data_glcd( cs, data & ~(1 << bit) );   // 読出したデータの 1 ドットを白く
    }
    while (1);
}
```

実習 4.2.2b グラフィック LCD にドット（黒 or 白）を描く関数を作成する

[解説]

ライブラリー "mt_e507.c" に void dot_glcd(int x, int y, int col) 関数を追加作成します。但し、(x,y）座標（左下隅が（0,0)、右上隅が（+127,+63))、col: 色（1: 黒 /0: 白）とします。合わせて "mt_e507.h" にその関数のプロトタイプ宣言を追加作成します。

[解答]

リスト 4.2.2b にプログラム例を示します。

リスト 4.2.2b　グラフィック LCD にドット（黒 or 白）を描く関数を作成する

```
/** main.c  ********************************************************************************/
#include    <mt_r300.h>                                       // マイコンボード用ライブラリー
#include    <mt_e507.h>                                       // グラフィック LCD ボード用ライブラリー
#define  CS2     0x01                                         // 右チップ (CS2) 選択 (CS2=0,CS1=1)
void fill_glcd_cs2()                                          // 画面の右半分 (CS2) を塗りつぶす関数
{
    int page, clmn;
    for (page = 0; page < 8; page++) {                        // ページ (0 ～ 7)
        set_page(CS2, page);                                  // ページセット
        set_clmn(CS2, 0);                                     // カラムセット ( 初期値 :0)
        for (clmn = 0; clmn < 64; clmn++) {                   // カラム (0 ～ 63)( 自動インクリメント )
            write_data_glcd(CS2, 0xff);                       // ディスプレイデータ ( 黒 ) の書き込み
        }
    }
}
void    main()
{
    initialize_glcd();                                        // GLCD の初期化
    wait_ms(500);
    fill_glcd_cs2();                                          // 画面の右半分を黒く塗りつぶす
    wait_ms(500);
    dot_glcd(  0, 0, 1);                                      // 座標 (x,y) にドット ( 黒 ) を描く
    dot_glcd( 63, 0, 1);                                      // 座標 (x,y) にドット ( 黒 ) を描く
    dot_glcd( 64, 0, 0);                                      // 座標 (x,y) にドット ( 白 ) を描く
    dot_glcd(127, 0, 0);                                      // 座標 (x,y) にドット ( 白 ) を描く
    dot_glcd(  0, 63, 1);                                     // 座標 (x,y) にドット ( 黒 ) を描く
    dot_glcd( 63, 63, 1);                                     // 座標 (x,y) にドット ( 黒 ) を描く
    dot_glcd( 64, 63, 0);                                     // 座標 (x,y) にドット ( 白 ) を描く
    dot_glcd(127, 63, 0);                                     // 座標 (x,y) にドット ( 白 ) を描く
    dot_glcd( -1, 10, 1);                                     // 無効座標
```

```
    dot_glcd( 10, -1, 1);                              // 無効座標
    dot_glcd(128, 10, 0);                              // 無効座標
    dot_glcd(120, 64, 0);                              // 無効座標
    while (1);
}
```

```
/** mt_e507.c ( 以下を追加します ) **********************************************************/
// 1 ドットを描く関数((x,y)座標(左下隅が(0,0),右上隅が(+127,+63)), col:色(1:黒/0:白))
void dot_glcd(int x, int y, int col)
{
    int cs, page, bit, clmn;
    char data;
    if( (0 <= x && x < 128) && (0 <= y && y < 64) )    // 有効座標?((0,0)～(+127,+63))
    {
        cs = 2 - x / 64;                               // チップセレクト(左:CS1(2)/右:CS2(1))
        clmn = x % 64;                                 // カラム
        page = (63 - y) / 8;                           // ページ
        bit = (63 - y) % 8;                            // ビット
        set_page( cs, page );                          // ページセット
        set_clmn( cs, clmn );                          // カラムセット
        read_data_glcd( cs );                          // DDRAM データ(ダミー)を読み出す
        data = read_data_glcd( cs );                   // DDRAM データ(有効)を読み出す
        set_clmn( cs, clmn );                          // カラム再セット(自動インクリメント)
        if( col )                                      // カラー?(1:黒/0:白)
            write_data_glcd( cs, data | (1 << bit) );  // 読出したデータの 1 ドットを黒く
        else                                           // 1<<bit = 1<<4 = 0x10
            write_data_glcd( cs, data & ~(1 << bit) ); // 読出したデータの 1 ドットを白く
    }
}
```

```
/** mt_e507.h ( 以下を追加します ) **********************************************************/
void    dot_glcd( int , int, int );
```

実習 4.2.3a グラフィック LCD に直線（連続したドットの集まり）を描く

[解説]

ここでは、既に作成した 1 ドットを描く関数を用いて直線を連続したドットの集まりで描きます。縦線、横線を簡単に描けますが、斜線は数学的な計算が必要になります。図 4-4 は直線の方程式の一例を示します。(5,5) - (63,34) 間の直線は、傾きが 1/2 で (5,5) 座標を通る直線となります。

その方程式は $y = (34 - 5)/(63 - 5) \cdot (x - 5) + 5$ となります。

従って、for() 文で x を 5 ～ 63 で連続的に変化させます。傾きが 45 度以上の場合は y の値を連続的に変化させます。これを逆にするとドットとドットの間が離れます。
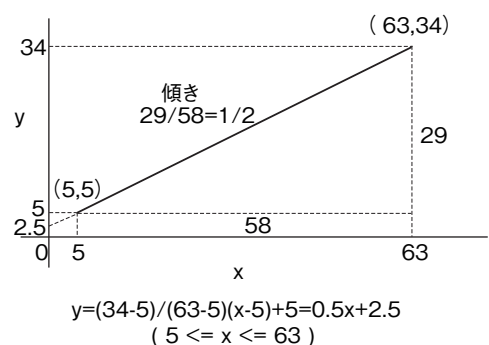
図 4-4　直線の方程式の一例

[解答]

リスト 4.2.3a にプログラム例を示します。

リスト 4.2.3a　グラフィック LCD に直線（連続したドットの集まり）を描く

```
/** main.c  *********************************************************************************/
#include    <mt_r300.h>                                    // マイコンボード用ライブラリー
#include    <mt_e507.h>                                    // グラフィック LCD ボード用ライブラリー
#define  CS2     0x01                                      // 右チップ (CS2) 選択 (CS2=0,CS1=1)
void fill_glcd_cs2()                                       // 画面の右半分 (CS2) を塗りつぶす関数
{
    // 省略（前方の同じ関数をコピー・ペーストして下さい）
}
void    main()
{
    int x, y;
    initialize_glcd();                                     // GLCD の初期化
    fill_glcd_cs2();                                       // 画面の右半分を黒く塗りつぶす
    wait_ms(100);
    // 横線 ((0,5)-(63,5), 黒 )
    for(x=0; x<=63; x++){
        dot_glcd( x, 5, 1);
        wait_ms(10);
    }
    // 縦線 ((5,0)-(5,63), 黒 )
    for(y=0; y<=63; y++){
        dot_glcd( 5, y, 1);
        wait_ms(10);
    }
    // 斜線 ((5,5)-(63,34), 黒 )
    for(x=5; x<=63; x++){                                  // 傾きが 45 度以下の時は x を制御する
        y = ((float)(34-5)/(float)(63-5))*(x-5)+5;
        dot_glcd( x, y, 1);
        wait_ms(10);
    }
    // 斜線 ((5,5)-(34,63), 黒 )
    for(x=5; x<=63; x++){                                  // 傾きが 45 度超の時に x を制御すると点線になる
        y = ((float)(63-5)/(float)(34-5))*(x-5)+5;
        dot_glcd( x, y, 1);
        wait_ms(10);
    }

    // 横線 ((64,5)-(127,5), 白 )
    for(x=64; x<=127; x++){
        dot_glcd( x, 5, 0);
        wait_ms(10);
```

```
    }
    // 縦線((69,0)-(69,63),白)
    for(y=0; y<=63; y++){
        dot_glcd( 69, y, 0);
        wait_ms(10);
    }
    // 斜線((69,5)-(98,63),白)
    for (y = 5; y<=63; y++){                    // 傾きが 45 度超の時は y を制御する
        x = ((float)(98-69)/(float)(63-5))*(y-5)+69;
        dot_glcd( x, y, 0);
        wait_ms(10);
    }
    // 斜線((69,5)-(127,34),白)
    for (y = 5; y<=34; y++){                    // 傾きが 45 度以下の時に y を制御すると点線になる
        x = ((float)(127-69)/(float)(34-5))*(y-5)+69;
        dot_glcd( x, y, 0);
        wait_ms(10);
    }
    while (1);
}
```

実習 4.2.3b グラフィック LCD に直線 (黒 or 白) を描く関数を作成する

[解説]

ライブラリー "mt_e507.c" に直線を描く関数 void line_glcd(int x1, int y1, int x2, int y2, int col) を追加作成し、"mt_e507.h" にその関数のプロトタイプ宣言を追加作成します。但し、(x1,y1) − (x2,y2) : 線分両端の座標、col: 色 (1: 黒/0: 白) とします。この内で abs() 関数を用いるため、ビルドで標準ライブラリーに stdlib.h (標準処理用ライブラリー) を組み込み、プログラムの先頭部に #include　<stdlib.h> を追加します。

この関数は連続したドットの集まりで直線を描きます。x1=x2、y1=y2 の場合、直線ではなく 1 点を描きます。直線の傾きが 45 度以下の場合は、x の値を連続的に変化させてドットを描きます。傾きが 45 度を超える場合は、y の値を連続的に変化させてドットを描きます。for() 文の制御変数の初期値と最終値の大小関係を調べ、逆順の場合はそれらを入れ替えます。

[解答]

リスト 4.2.3b にプログラム例を示します。

リスト 4.2.3b　グラフィック LCD に直線 (黒 or 白) を描く関数を作成する

```
/** main.c  ******************************************************************************/
#include    <mt_r300.h>                         // マイコンボード用ライブラリー
#include    <mt_e507.h>                         // グラフィック LCD ボード用ライブラリー
#define  CS2     0x01                           // 右チップ (CS2) 選択 (CS2=0,CS1=1)
void fill_glcd_cs2()                            // 画面の右半分 (CS2) を塗りつぶす関数
{
    // 省略 (前方の同じ関数をコピー・ペーストして下さい)
}
void main()
```

```
{
    initialize_glcd();                          // GLCD の初期化
    fill_glcd_cs2();                            // 画面の右半分を黒く塗りつぶす
    wait_ms(100);
    line_glcd(  0,  5, 63,  5, 1);              // 横線 ((0,5)-(63,5), 黒 )
//  line_glcd( 63,  5,  0,  5, 1);              // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd(  5,  0,  5, 63, 1);              // 縦線 ((5,0)-(5,63), 黒 )
//  line_glcd(  5, 63,  5,  0, 1);              // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd(  5,  5, 63, 34, 1);              // 斜線 ((5,5)-(63,34), 黒 )
//  line_glcd( 63, 34,  5,  5, 1);              // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd(  5,  5, 34, 63, 1);              // 斜線 ((5,5)-(34,63), 黒 )
//  line_glcd( 34, 63,  5,  5, 1);              // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd( -5, -5, 68, 68, 1);              // 斜線 ((-5,-5)-(68,68), 黒 )
//  line_glcd( 68, 68, -5, -5, 1);              // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd( 34, 10, 34, 10, 1);              // 点 ( 黒 )
    wait_ms(500);

    line_glcd(  64,  5,127,  5, 0);             // 横線 ((64,5)-(127,5), 白 )
//  line_glcd( 127,  5, 64,  5, 0);             // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd(  69,  0, 69, 63, 0);             // 縦線 ((69,0)-(69,63), 白 )
//  line_glcd(  69, 63, 69,  0, 0);             // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd(  69,  5, 98, 63, 0);             // 斜線 ((69,5)-(98,63), 白 )
//  line_glcd(  98, 63, 69,  5, 0);             // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd(  69,  5,127, 34, 0);             // 斜線 ((69,5)-(127,34), 白 )
//  line_glcd( 127, 34, 69,  5, 0);             // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd(  59, -5,132, 68, 0);             // 斜線 ((59,-5)-(132,68), 白 )
//  line_glcd( 132, 68, 59, -5, 0);             // 上行の 2 点を逆順に指定して確認する
    wait_ms(200);
    line_glcd( 98,  10, 98, 10, 0);             // 点 ( 白 )
    while (1);
}
```

```
/** mt_e507.c ( 以下を追加します ) ********************************************************/
#include    <stdlib.h>                          // 標準処理用ライブラリー ( プログラムの先頭に移動 )
// 直線を描く関数 ((x1,y1)-(x2,y2): 対角座標 ,col:(1: 黒 /0: 白 ))
void line_glcd(int x1, int y1, int x2, int y2, int col)
```

```
{
    int x, xd1, xd2, y, yd1, yd2, w;
    if( abs(y2-y1) <= abs(x2-x1) )                  // ⊿y <= ⊿x ?
    {                                               // 傾き 45 度以下の時
        if((x2-x1) == 0)                            // ⊿x = 0 の時 , 当然 ⊿y = 0
            dot_glcd( x1, y1, col );                // 2 点が重なるため 1 点を描く
        else
        {
            if(x1 > x2){                            // x1<x2 となるように入れ替える
                w = x2,      x2 = x1, x1 = w;
                w = y2,      y2 = y1, y1 = w;
            }
            xd1 = ( x1 < 0 )? 0 : x1;               // 0 以上
            xd2 = ( 127 < x2 )? 127 : x2;           // 127 以下
            for (x = xd1; x <= xd2; x++){           // x を制御して直線を描く
                y = ((float)(y2-y1)/(float)(x2-x1))*(x-x1)+y1;
                dot_glcd( x, y, col );
            }
        }
    }
    else                                            // 傾き 45 度を超える時
    {
        if(y1 > y2){                                // y1<y2 となるように入れ替える
            w = x2, x2 = x1, x1 = w;
            w = y2, y2 = y1, y1 = w;
        }
        yd1 = ( y1 < 0 )? 0 : y1;                   // 0 以上
        yd2 = ( 63 < y2 )? 63 : y2;                 // 63 以下
        for (y = yd1; y <= yd2; y++){               // y を制御して直線を描く
            x = ((float)(x2-x1)/(float)(y2-y1))*(y-y1)+x1;
            dot_glcd( x, y, col );
        }
    }
}
```

```
/** mt_e507.h ( 以下を追加します ) ***********************************************************/
void    line_glcd( int , int , int , int , int );
```

実習 4.2.4 グラフィック LCD に四角形を描く関数を作成する

[解説]

ライブラリー "mt_e507.c" に四角形を描く関数 void rectangle_glcd(int x1, int y1, int x2, int y2, int col, int fill) を追加作成し、"mt_e507.h" にその関数のプロトタイプ宣言を追加作成します。但し、(x1,y1) − (x2,y2): 対角座標、col: 色 (1: 黒 /0: 白)、fill: 塗潰し (1: 有 /0: 無) とします。

この関数は塗り潰しの時には連続した直線の集まりで四角形を描き、塗り潰さない時には 4 本の外枠線で四角形を描きます。これは 1 ドット単位で描きます。for() 文の制御変数の初期値と最終値の大小関係を調べ、逆順の場合

はそれらを入れ替えます。

[解答]

リスト 4.2.4 にプログラム例を示します。

リスト 4.2.4　グラフィック LCD に四角形を描く関数を作成する (1 ドット単位で)

```
/** main.c  *******************************************************************************/
#include    <mt_r300.h>                          // マイコンボード用ライブラリー
#include    <mt_e507.h>                          // グラフィック LCD ボード用ライブラリー
void main()
{
    initialize_glcd();                           // GLCD の初期化
    wait_ms(500);
    rectangle_glcd( 3,  3, 124, 60, 1, 0);       // (x1,y1)-(x2,y2) 対角に ,1: 黒で ,0: 塗り潰さない
    wait_ms(500);
    rectangle_glcd( 5,  5, 122, 58, 1, 1);       // (x1,y1)-(x2,y2) 対角に ,1: 黒で ,1: 塗り潰す
//  rectangle_glcd( 5, 58, 122,  5, 1, 1);       // 上行の対角座標を入れ替えて確認する
//  rectangle_glcd( 122, 5,  5, 58, 1, 1);       // 上行の対角座標を入れ替えて確認する
//  rectangle_glcd( 122, 58, 5,  5, 1, 1);       // 上行の対角座標を入れ替えて確認する
    wait_ms(500);
    rectangle_glcd(25, 15, 102, 48, 0, 1);       // (x1,y1)-(x2,y2) 対角に ,0: 白で ,1: 塗り潰す
    wait_ms(500);
    rectangle_glcd(26, 16, 101, 47, 1, 0);       // (x1,y1)-(x2,y2) 対角に ,1: 黒で ,0: 塗り潰さない
    wait_ms(500);
    rectangle_glcd(-5, -5, 63,  31, 1, 1);       // ( 画面外 )-(x2,y2) 対角に ,1: 黒で ,1: 塗り潰す
    wait_ms(500);
    rectangle_glcd(64, 32, 132, 68, 1, 0);       // (x1,y1)-( 画面外 ) 対角に ,1: 黒で ,0: 塗り潰さない
    while (1);
}
```

```
/** mt_e507.c ( 以下を追加します ) *********************************************************/
// 四角形を描く関数 ((x1,y1)-(x2,y2): 対角座標 ,col:(1: 黒 /0: 白 ),fill: 塗潰し (1: 有 /0: 無 ))
void rectangle_glcd(int x1, int y1, int x2, int y2, int col, int fill)
{
    int y, w;
    if(fill){        // 塗り潰す
        if(y1 > y2){
            w = y2, y2 = y1, y1 = w;             // y1<y2 となるように入れ替える
        }
        y1 = ( y1 < 0 )? 0 : y1;                 // 0 以上
        y2 = ( 63 < y2 )? 63 : y2;               // 63 以下
        for (y = y1; y <= y2; y++)
            line_glcd(x1, y, x2, y, col);
    }
    else{       // 塗り潰さない ( 枠線 )
```

```
        line_glcd(x1, y1, x2, y1, col);
        line_glcd(x1, y2, x2, y2, col);
        line_glcd(x1, y1, x1, y2, col);
        line_glcd(x2, y1, x2, y2, col);
    }
}
```

```
/** mt_e507.h ( 以下を追加します ) ***********************************************************/
void    rectangle_glcd( int, int, int, int, int, int );
```

この関数は直線を描く関数を利用して 1 ドット単位で描いているため処理速度が遅いです。8 ドット分をあらかじめ計算しておき、それを一括で描く関数を作成すれば処理速度が速くなります。このプログラムは若干複雑になるため応用編で解説します。

実習 4.2.5a グラフィック LCD に円やリサージュ図形を描く（sin θ、cos θを用いる）

[解説]

円の方程式は角度θ [rad] をパラメータとする次式で求めます。

$x=\cos\theta \qquad (0 \leqq \theta < 2\pi)$

$y=\sin\theta \qquad (0 \leqq \theta < 2\pi)$

この x、y は－ 1.0 ～ 0 ～ +1.0 の小さな値です。グラフィック LCD にこれを大きく表示するために、x,y に適切な演算（拡大、シフト）を行い、有効座標（$0 \leqq x \leqq 127, 0 \leqq y \leqq 63$）に収めます。

x の拡大率を 2 倍にすると横長の楕円になります。x の周波数を 2 倍にしたり、位相を 45° 増やしたりすると異なる波形が観測できます。縦横の倍率、周波数比、位相差を変えて種々の波形を観測しましょう。これらをリサージュ図形といいます。

[解答]

リスト 4.2.5a にプログラム例を示します。

リスト 4.2.5a　グラフィック LCD に円やリサージュ図形を描く（sin θ、cos θ を用いる）

```
/** main.c  ******************************************************************************/
#include    <mathf.h>                        // 数値計算用ライブラリー ( 標準ライブラリーも組み込みます )
#include    <mt_r300.h>                      // マイコンボード用ライブラリー
#include    <mt_e507.h>                      // グラフィック LCD ボード用ライブラリー
#define  PI  3.1415927                       // 単精度浮動小数点数 (7.2 桁 )
void main()
{
    float    xf, yf;
    int      x, y, k;
    initialize_glcd();                               // GLCD の初期化
    line_glcd( 64, 0, 64, 63, 1);
    line_glcd( 0, 32, 127, 32, 1);
    for (k = 0; k < 360; k++){                       // 角度制御
        xf = cosf( PI / 180 * k );                   // 円
```

```
//      xf = 2 * cosf( PI / 180 * k);              // 楕円（x 方向を 2 倍に拡大する）
//      xf = 2 * cosf( PI / 180 * 2 * k);          // リサージュ図形（x2 倍，周波数 2 倍）
//      xf = 2 * cosf( PI / 180 * (2 * k + 45)); // リサージュ図形（x2 倍，周波数 2 倍，位相差 45°）
//      縦横の倍率、周波数比、位相差を変えて種々のリサージュ図形を観測しましょう
        yf = sinf( PI / 180 * k );
        x = 31.5 * xf + 63.5;                      // 円（半径 =31.5，中心座標（63.5, 31.5））
        y = 31.5 * yf + 31.5;
        dot_glcd(x, y, 1);
        wait_ms(10);
    }
    while (1);
}
```

実習 4.2.5b グラフィック LCD に円を描く関数を作成する（1 ドット単位で）

[解説]

直前の方法では三角関数を使わなければならないし、そのパラメータ（角度）の適切なきざみ値を求めることが難しいです。円の塗り潰しもできません。

ここでは次のようなもっと簡単な方法を用います。円を囲む四角形内の一つひとつのドットについて、円周上か、円の外側か、円の内側かを判別し、円（塗り潰し円も含む）を描きます。

図 4-5 は中心座標（x0,y0）、半径 r の円を示します。あるドット（x,y）の中心点からの距離は$\sqrt{(x-x0)^2+(y-y0)^2}$となります。それが r より大きければ円の外側、一致したら円周上、小さければ円の内側となります。四角内のすべてのドットについてこの判別をします。浮動小数点数では完全に一致しないため整数型で判別します。若干の差を認めれば太い線になります。

円周上の点 (x,y)
$\sqrt{(x-x0)^2+(y-y0)^2} = r$
円の外側の点 (x,y)
$\sqrt{(x-x0)^2+(y-y0)^2} > r$
円の内側の点 (x,y)
$\sqrt{(x-x0)^2+(y-y0)^2} < r$
(x0,y0)
中心座標
r: 半径
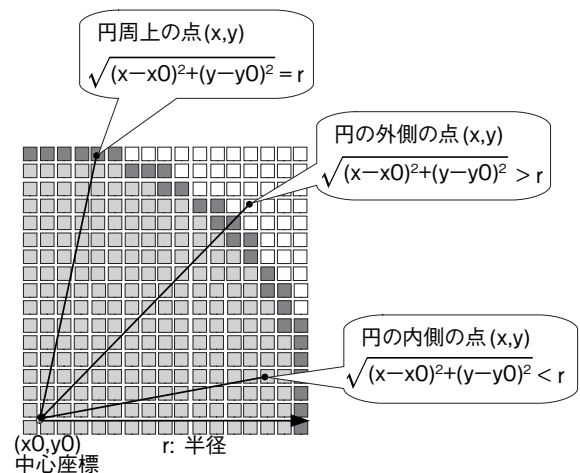
図 4-5　円の内外の判定

[解答]

リスト 4.2.5b にプログラム例を示します。

リスト 4.2.5b　グラフィック LCD に円を描く関数を作成する（1 ドット単位で）

```
/** main.c  ******************************************************************************/
#include    <mt_r300.h>                            // マイコンボード用ライブラリー
#include    <mt_e507.h>                            // グラフィック LCD ボード用ライブラリー
void main()
{
    initialize_glcd();                             // GLCD の初期化
    wait_ms(500);
    circle_glcd(32, 32, 30, 1, 0);                 // 中心座標（x0,y0），半径 r，黒色，塗り潰さない
    wait_ms(500);
    circle_glcd(25, 25, 20, 1, 0);
    wait_ms(500);
```

```
    circle_glcd(18, 18, 10, 1, 0);
    wait_ms(500);
    circle_glcd(96, 32, 30, 1, 1);                  // 中心座標 (x0,y0), 半径 r, 黒色 , 塗り潰す
//  circle_glcd(96, 32, 30, 1, 0);                  // 中心座標 (x0,y0), 半径 r, 黒色 , 塗り潰す
    wait_ms(500);
    circle_glcd(103, 25, 20, 0, 1);                 // 中心座標 (x0,y0), 半径 r, 白色 , 塗り潰す
//  circle_glcd(103, 25, 20, 1, 0);                 // 中心座標 (x0,y0), 半径 r, 白色 , 塗り潰す
    wait_ms(500);
    circle_glcd(110, 18, 10, 1, 1);                 // 中心座標 (x0,y0), 半径 r, 黒色 , 塗り潰す
//  circle_glcd(110, 18, 10, 1, 0);                 // 中心座標 (x0,y0), 半径 r, 黒色 , 塗り潰す
    wait_ms(500);
    circle_glcd(-5, -5, 19, 1, 0);                  // 画面外の中心座標
    wait_ms(500);
    circle_glcd(133, -5, 19, 1, 1);                 // 画面外の中心座標
    while (1);
}
```

```
/** mt_e507.c ( 以下を追加します ) ********************************************************/
//#include  <mathf.h>                               // 数値計算用ライブラリー ( プログラムの先頭に移動 )
void circle_glcd(int x0, int y0, int r, int col, int fill)
{
    int x, x1, x2, y, y1, y2;
    x1 = ( x0 - r < 0 )?  0 : x0 - r;               // 0 以上
    x2 = ( 127 < x0 + r )? 127 : x0 + r;            // 127 以下
    y1 = ( y0 - r < 0 )?  0 : y0 - r;               // 0 以上
    y2 = ( 63 < y0 + r )? 63 : y0 + r;              // 63 以下
    if(fill){                                       // 塗り潰す
        for (y = y1; y <= y2; y++)
            for (x = x1; x <= x2; x++)
                if((int)r - (int)sqrtf((x-x0)*(x-x0)+(y-y0)*(y-y0)) >= 0) // 円と塗り潰し円
                    dot_glcd( x, y, col);
    }
    else{                                           // 塗り潰さない
        for (y = y1; y <= y2; y++)
            for (x = x1; x <= x2; x++)
                if((int)r - (int)sqrtf((x-x0)*(x-x0)+(y-y0)*(y-y0)) == 0)     // 円 ( 細線 )
//              if(abs((int)r - (int)sqrtf((x-x0)*(x-x0)+(y-y0)*(y-y0))) <= 1)// 円 ( 太線 )
                    dot_glcd( x, y, col);
    }
}
```

```
/** mt_e507.h ( 以下を追加します ) **********************************************************/
void    circle_glcd( int, int, int, int, int);
```

この関数も 1 ドット単位で描いているため処理速度が遅いです。8 ドット分をあらかじめ計算し、一括で描く関数

を作成すれば処理速度が速くなります。このプログラムも応用編で解説します。

実習 4.2.6 グラフィック LCD にグラフ（正弦波形など）を描く

[解説]

x を 0 ～ 127 で変化させ、xf を 0 ～ 2 π とします。その正弦波、余弦波を表示します。

[解答]

リスト 4.2.6 にプログラム例を示します。

リスト 4.2.6　グラフィック LCD にグラフ（正弦波形など）を描く

```
/** main.c  ***************************************************************************/
#include    <mathf.h>                           // 数値計算用ライブラリー
#include    <mt_r300.h>                         // マイコンボード用ライブラリー
#include    <mt_e507.h>                         // グラフィック LCD ボード用ライブラリー
#define  PI  3.1415927                          // 単精度浮動小数点数（7.2 桁）
void main()
{
    float    xf, yf, y2f;
    int      x, y, y2;
    initialize_glcd();                          // GLCD の初期化
    line_glcd( 0, 0, 0, 63, 1);                 // 垂直軸
    line_glcd( 0, 32, 127, 32, 1);              // 時間 ( 水平 ) 軸
    for (x = 0; x < 128; x++){                  // x=0 ～ 127
        xf = 2 * PI * x / 127.0;                // xf=0 ～ 2 π
        yf = sinf(xf);                          // yf=-1.0 ～ +1.0
        y = 31.5 * yf + 31.5;                   // y=0 ～ 63
        dot_glcd( x, y, 1);
        y2f = cosf(xf);                         // y2f=-1.0 ～ +1.0
        y2 = 31.5 * y2f + 31.5;                 // y2=0 ～ 63
        dot_glcd( x, y2, 1);
        wait_ms(10);
    }
    while (1);
}
```

実習 4.2.7a グラフィック LCD にイメージを描く（ページとカラムを指定する）

[解説]

ページ、幅、配列データを持ったイメージの構造体を作成します。

[解答]

リスト 4.2.7a にプログラム例を示します。

リスト 4.2.7a　グラフィック LCD にイメージを描く（ページとカラムを指定する）

```
/** main.c  ***************************************************************************/
#include    <mt_r300.h>                                 // マイコンボード用ライブラリー
```

```
#include    <mt_e507.h>                              // グラフィック LCD ボード用ライブラリー
struct struct_pwimage                               // イメージの構造体
{
    int p;                                          // ページ
    int w;                                          // 幅
    char *img;                                      // 配列データ
};
char cap_a[] = {0x7e,0x11,0x11,0x11,0x7e};          // 大文字の 'A'( 高さ (7)* 幅 (5)= ページ (1)* 幅 (5))
char cap_b[] = {0x7f,0x49,0x49,0x49,0x36};          // 大文字の 'B'( 高さ (7)* 幅 (5)= ページ (1)* 幅 (5))
                                                    // D0    ■■■      ■■■■
                                                    // D1  ■      ■   ■      ■
                                                    // D2  ■      ■   ■      ■
                                                    // D3  ■      ■   ■■■■
                                                    // D4  ■■■■■   ■      ■
                                                    // D5  ■      ■   ■      ■
                                                    // D6  ■      ■   ■■■■
                                                    // D7
char kan[] = // 全角文字の ' 漢 '( 高さ (16)* 幅 (16)= ページ (2)* 幅 (16))
   {0x22,0x66,0xcc,0x00,0x00,0x02,0xf2,0x92,0x97,0x92,0xf2,0x92,0x97,0x92,0xf2,0x02,
    0x80,0xe0,0x38,0x0e,0x00,0x88,0x8a,0xca,0x6a,0x3a,0x0f,0x3a,0x6a,0xca,0x8a,0x88};
char ji[] = // 全角文字の ' 字 '( 高さ (16)* 幅 (16)= ページ (2)* 幅 (16))
   {0x00,0x1c,0x04,0x24,0x24,0x24,0x24,0x27,0x24,0xa4,0xe4,0x64,0x24,0x04,0x1c,0x00,
    0x02,0x02,0x02,0x02,0x02,0x82,0x82,0xff,0x03,0x02,0x02,0x02,0x02,0x02,0x02,0x02};
char logo[] = // サンハヤトのロゴマーク ( 高さ (24)* 幅 (24)= ページ (3)* 幅 (24))
   {0x00,0xc0,0x70,0x30,0x3c,0x3e,0x7e,0x7e,0x7f,0x7f,0xff,0xff,                  // page0
    0xff,0xff,0xff,0xfe,0xfe,0xfe,0xfc,0xfc,0xf0,0xe0,0xc0,0x00,
    0x3e,0xff,0xff,0xff,0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x00,0xe0,                  // page1
    0xf0,0xf1,0xf9,0xf9,0xfb,0xfb,0xfb,0xf7,0xf7,0xef,0xef,0x1e,
    0x00,0x01,0x07,0x0f,0x1f,0x3f,0x3f,0x7f,0xff,0xff,0xff,0xfc,                  // page2
    0xff,0xff,0xff,0xff,0x7f,0x3f,0x3f,0x1f,0x0f,0x07,0x01,0x00};
struct struct_pwimage st_cap_a = { 1, 5, cap_a};
struct struct_pwimage st_cap_b = { 1, 5, cap_b};
struct struct_pwimage st_kan    = { 2,16, kan};
struct struct_pwimage st_ji     = { 2,16, ji};
struct struct_pwimage st_logo   = { 3, 24, logo};
void putpwimage_glcd( int page1, int x1, struct struct_pwimage *struct_img )
{
    int x, x2, cs, clmn, page, page2;
    char dot_dt;
    page = (page1 < 0)? 0 : page1;                                                  // 起点
    page2 = ((page1 + struct_img->p - 1) < 8)? (page1 + struct_img->p - 1) : 7;     // 終点
    while(page <= page2){
        x = (x1 < 0)? 0 : x1;                                                       // 起点
        x2 = ((x1 + struct_img->w - 1) < 128)? (x1 + struct_img->w - 1) : 127;      // 終点
        while(x <= x2){
```

```
            cs = 2 - x / 64;                          // チップセレクト ( 左 :CS1(2)/ 右 :CS2(1))
            clmn = x % 64;                            // カラム
            set_page(cs, page);
            set_clmn(cs, clmn);
            read_data_glcd( cs );                     // DDRAM データ ( ダミー ) を読み出す
            dot_dt = read_data_glcd( cs );            // DDRAM データ ( 正常 ) を読み出しバッファに格納
            set_clmn(cs, clmn);
            write_data_glcd(cs, struct_img->img[(page-page1)*struct_img->w+x-x1] | dot_dt);
            x++;
        }
        page++;
    }
}
void main()
{
    int page;
    initialize_glcd();                                        // GLCD の初期化
    putpwimage_glcd( 0, 10, &st_cap_a );
    wait_ms(300);
    putpwimage_glcd( 0, 16, &st_cap_b );
    wait_ms(300);
    putpwimage_glcd( 2, 30, &st_kan );
    wait_ms(300);
    putpwimage_glcd( 2, 47, &st_ji );
//  putpwimage_glcd( 2, 55, &st_ji );                         // " 字 " の重なり確認
    wait_ms(300);
    putpwimage_glcd( 5, 80, &st_logo );
    wait_ms(1000);
    for(page=-3;page<=8;page++){                              // 範囲外も確認
        clear_glcd();
        putpwimage_glcd( page, 16 * page, &st_logo );     // イメージの移動
        wait_ms(300);
    }
    while (1);
}
```

実習 4.2.7b グラフィック LCD にイメージを描く関数を作成する（ページとカラムを指定する）

[解説]

ライブラリー "mt_e507.c" にイメージを描く関数 void putpwimage_glcd(int page1, int x1, struct struct_pwimage *struct_img) を追加作成し、"mt_e507.h" にそのプロトタイプ宣言を追加作成します。但し、(page1,x1)：イメージ左上隅座標（page1:0 ～ 7,x1:0 ～ 127)、*struct_img: 構造体とします。

[解答]

リスト 4.2.7b にプログラム例を示します。

リスト 4.2.7b　グラフィック LCD にイメージを描く関数を作成する (ページとカラムを指定する)

```
/** main.c  ***************************************************************************/
#include    <mt_r300.h>                                         // マイコンボード用ライブラリー
#include    <mt_e507.h>                                         // グラフィック LCD ボード用ライブラリー
struct struct_pwimage                                           // イメージの構造体
{
    int p;                                                      // ページ
    int w;                                                      // 幅
    char *img;                                                  // 配列データ
};
// char cap_a[]= 〜 char logo[]= は直前の実習と同じです ( その部分をコピー・ペーストします )
struct struct_pwimage st_cap_a = { 1, 5, cap_a};
struct struct_pwimage st_cap_b = { 1, 5, cap_b};
struct struct_pwimage st_kan    = { 2,16, kan};
struct struct_pwimage st_ji     = { 2,16, ji};
struct struct_pwimage st_logo   = { 3, 24, logo};
void main()
{
    int page;
    initialize_glcd();                                          // GLCD の初期化
    putpwimage_glcd( 0, 10, &st_cap_a );
    wait_ms(300);
    putpwimage_glcd( 0, 16, &st_cap_b );
    wait_ms(300);
    putpwimage_glcd( 2, 30, &st_kan );
    wait_ms(300);
    putpwimage_glcd( 2, 47, &st_ji );
//  putpwimage_glcd( 2, 55, &st_ji );                           // " 字 " の重なり確認
    wait_ms(300);
    putpwimage_glcd( 5, 80, &st_logo );
    wait_ms(1000);
    for(page=-3;page<=8;page++){                                // 範囲外も確認
        clear_glcd();
        putpwimage_glcd( page, 16 * page, &st_logo );           // イメージの移動
        wait_ms(300);
    }
    while (1);
}
```

```
/** mt_e507.c ( 以下を追加します ) **********************************************************/
struct struct_pwimage                                           // イメージの構造体
{
    int p;                                                      // ページ
    int w;                                                      // 幅
    char *img;                                                  // 配列データ
```

```
};
void putpwimage_glcd( int page1, int x1, struct struct_pwimage *struct_img )
{
    int x, x2, cs, clmn, page, page2;
    char dot_dt;
    page = (page1 < 0)? 0 : page1;                                                  // 起点
    page2 = ((page1 + struct_img->p - 1) < 8)? (page1 + struct_img->p - 1) : 7;    // 終点
    while(page <= page2){
        x = (x1 < 0)? 0 : x1;                                                       // 起点
        x2 = ((x1 + struct_img->w - 1) < 128)? (x1 + struct_img->w - 1) : 127;     // 終点
        while(x <= x2){
            cs = 2 - x / 64;                        // チップセレクト ( 左 :CS1(2)/ 右 :CS2(1))
            clmn = x % 64;                          // カラム
            set_page(cs, page);
            set_clmn(cs, clmn);
            read_data_glcd( cs );                   // DDRAM データ ( ダミー ) を読み出す
            dot_dt = read_data_glcd( cs );          // DDRAM データ ( 正常 ) を読み出しバッファに格納
            set_clmn(cs, clmn);
            write_data_glcd(cs, struct_img->img[(page-page1)*struct_img->w+x-x1] | dot_dt);
            x++;
        }
        page++;
    }
}
```

```
/** mt_e507.h ( 以下を追加します ) ***********************************************************/
void    putpwimage_glcd( int, int, struct struct_pwimage * );
```

この関数はページ(0 ～ 7 行)とカラム(0 ～ 127)を指定してイメージを描く関数です。行と行の間には描けません。そこで、(x,y) 座標を指定してイメージを描く関数を作成すれば自由な位置にイメージを描けるため便利です。このプログラムも若干複雑になるため応用編で解説します。

更に半角文字のフォントをあらかじめ作成し、(x,y) 座標を指定して文字を描く関数 void putchar_glcd(int x1, int y1, struct struct_image *st_img, unsigned char c) や文字列を描く関数 void prints_glcd(int x1, int y1, struct struct_image *st_img, char *str) なども応用編で解説します。

5. おわりに

この実習により、グラフィック LCD を制御する基本プログラムが理解できたことと思います。各信号のレベル（H/L）の変化を視認するモニター LED も効果的だったと思います。それらの制御プログラムをライブラリーにまとめましたので、今後はそれを活用して、種々の制御プログラムを作成しましょう。

なおブイポートラボでは、このトレーニングテキストの応用編として、途中省略したライブラリー関数や、他の基板（アナログ入出力基板など）と組み合わせたテキストを作成中です。

マイコントレーニング基板　MT-R300
トレーニングテキスト　グラフィック LCD 基板編

発行日　　2012-4-20　Rev1.00
発　行　　サンハヤト株式会社
　　　　　〒 170-0005　東京都豊島区南大塚 3 丁目 40 番 1 号

SG12005